2006

Primitives and Schemes for Non-Atomic Information Authentication

Goce Jakimoski

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

PRIMITIVES AND SCHEMES FOR NON-ATOMIC INFORMATION AUTHENTICATION

By

GOCE JAKIMOSKI

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Spring Semester, 2006

The members of the Committee approve the Dissertation of Goce Jakimoski defended on March 30, 2006.

Mike Burmester
Professor Directing Dissertation

Yvo Desmedt
Professor Co-Directing Dissertation

Mark Van Hoeij
Outside Committee Member

Kyle Gallivan
Committee Member

Michael Mascagni
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

ACKNOWLEDGEMENTS

The completion of the dissertation would not have been possible without the support of my family that has blessed my life in ways I cannot repay. I wish to recognize, with profound appreciation, the invaluable academic mentoring I received from my advisors: Professor Yvo Desmedt and Professor Mike Burmester. In addition to my advisors, my sincere thanks are due to the members of my advisory committee, Dr. Mark Van Hoeij, Dr. Kyle Gallivan and Dr. Michael Mascagni, whose guidance and encouragement contributed to the completion of this work. I would also like to thank the Department of Computer Science for giving me an opportunity and providing an environment where I can study and conduct my research. My final thanks goes to the National Science Foundation (NSF) for the financial help through a number of grants awarded to my academic advisors. — Goce

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The digital revolution, fired by the development of the information and communication technologies, has fundamentally changed the way we think, behave, communicate, work and earn livelihood (the World Summit on the Information Society). These technologies have affected all aspects of our society and economy. However, the Information Society developments present us not only with new benefits and opportunities, but also with new challenges. Information security is one of these challenges, and nowadays, information security mechanisms are inevitable components of virtually every information system.

Information authentication is one of the basic information security goals, and it addresses the issues of source corroboration and improper or unauthorized modification of data. More specific, data integrity is the property that the data has not been changed in an unauthorized manner since its creation, transmission or storage. Data origin authentication, or message authentication, is the property whereby a party can be corroborated as a source of the data.

Usually, message authentication is achieved by appending an authentication tag or a digital signature to the message. The authentication tag (resp., digital signature) is computed in such a way so that only an entity that is in possession of the secret key can produce it, and it is used by the verifier to determine the authenticity of the message. During this procedure, the message is considered to be an atomic object in the following sense. The verifier needs the complete message in order to check its validity. Presented with the authentication tag (resp., digital signature) and an incomplete message, the verifier cannot determine whether the presented incomplete message is authentic or not. We consider a more general authentication model, where the verifier is able to check the validity of incomplete messages. In particular, we analyze the cases of erasure-tolerant information authentication and stream authentication.

Our model of erasure-tolerant information authentication assumes that a limited number of the message "letters" can be lost during the transmission. Nevertheless, the verifier should still be able to check the authenticity of the received incomplete message. We provide answers to several fundamental questions in this model (e.g., lower bounds on the deception probability, distance properties, optimal constructions, etc.), and we propose some constructions of erasure-tolerant authentication codes.

Streams of data are bit sequences of a finite, but a priori unknown length that a sender sends to one or more recipients, and they occur naturally when on-line processing is required. In this case, the receiver should be able to verify the authenticity of a prefix of the stream, that is, the part of the stream that has been received so far. We provide efficient and proven secure schemes for both unicast and multicast stream authentication. The security proof of one of the proposed multicast stream authentication schemes assumes that the underlying block cipher is a related-key secure pseudorandom permutation. So, we also study the resistance of AES (Advanced Encryption Standard) to related-key differential attacks.

CHAPTER 1

Introduction

This chapter reviews some of the elementary information authentication notions and solutions. We also briefly describe the contributions of the dissertation.

1.1 Background

1.1.1 Block Ciphers

Block encryption algorithms (or *block ciphers*) are efficient algorithms that take as input a key and a plaintext block (i.e., a bit string of fixed size) and output a ciphertext block. Usually, the length of the output block is equal to the length of the input block. With the current state of art, the design of block ciphers is heuristic, and it is assumed that a block cipher is secure if there is no known attack that can break it faster than the exhaustive search attack. There are many constructions (e.g., encryption schemes, message authentication schemes, pseudorandom generators and hash functions) whose security relies on the security of the underlying block cipher. In this section, we briefly describe the Advanced Encryption Standard and some general attacks on block encryption algorithms.

Advanced Encryption Standard

On October 2, 2000, after a long and complex evaluation process, NIST announced that it has selected Rijndael to propose for the Advanced Encryption Standard. A draft standard was published for public review and comment, and in 2001, FIPS-197 was approved as a Federal Information Processing Standard for the AES. The algorithm is being used by the U.S. Government, and on a voluntary basis, by the private sector. The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits. It may be used with three different key lengths (128, 192, and 256 bits), and these different "flavors" are referred to as "AES-128", "AES-192", and "AES-256". A brief description of the encryption and decryption algorithms is given below.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
     byte state[4,Nb]

     state = in

     AddRoundKey(state, w[0, Nb-1])

     for round = 1 step 1 to Nr1
         SubBytes(state)
         ShiftRows(state)
         MixColumns(state)
         AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
     end for

     SubBytes(state)
     ShiftRows(state)
     AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

     out = state
end
```

Figure 1.1: Pseudocode for AES encryption

Usually, block ciphers are iterative ciphers, that is, some transformation, which is called an *encryption round*, is iteratively applied to the input to derive the output. The key that is used by an encryption round is called a *round key*, and the only difference between two different encryption rounds is that they used different round keys.

The design of AES is based on the same paradigm with a small exception that the final round is slightly different than the previous rounds. A pseudocode that describes the AES encryption process is given in Fig 1.1. The meaning of the parameters is the following. Nb is the number of 32-bit words in a block, and it is fixed (Nb=4) for all AES variants. Nr is the number of rounds and it is different for different key lengths (i.e., Nr=10 for AES-128, Nr=12 for AES-192 and Nr=14 for AES-256). The byte array in stores the input block. The key is given in the word array w, and the output is returned in the byte array out. The AES encryption round consists of four invertible transformations: AddRoundKey, SubBytes, ShiftRows and MixColumns. In the final round, the MixColumns transformation is omitted since it does not contribute to the security of the

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
     word temp

     i = 0

     while (i < Nk)
          w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
          i = i+1
     end while

     i = Nk

     while (i < Nb * (Nr+1)]
          temp = w[i-1]
          if (i mod Nk = 0)
               temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
          else if (Nk > 6 and i mod Nk = 4)
               temp = SubWord(temp)
          end if
          w[i] = w[i-Nk] xor temp
          i = i + 1
     end while
end
```

Figure 1.2: Pseudocode for the key expansion

cipher.

The AddRoundKey transformation is simply a bitwise XOR of the round key and the array *state*. The round keys are derived from a small randomly selected secret key using a key scheduling algorithm. The AES key scheduling algorithm is depicted in Fig 1.2.

SubBytes is a non-linear byte substitution that transforms each byte independently using a substitution table (S-box). The function defined by the substitution table is a bijection, and it is constructed by composing two transformations: first, a multiplicative inverse is taken in a finite field GF(2^8) (the additive identity element is mapped to itself), and then, an affine transformation is applied to get the output byte. The S-box is given in Table 1.1.

ShiftRows permutes the bytes of the state. The state can be represented as a 4×4 array of bytes, where the first word of the block corresponds to the first column, the second word corresponds to

Table 1.1: AES S-box: the substitution values of the byte xy

x/y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

the second column, etc. ShiftRows cyclically shifts the bytes of the last three columns of the state by one, two and three bytes correspondingly. The resultant permutation is

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 5 & 10 & 15 & 4 & 9 & 14 & 3 & 8 & 13 & 2 & 7 & 12 & 1 & 6 & 11 \end{pmatrix}.$$

MixColumns applies a linear transformation to each column of the state represented as a 4×4 array of bytes. Each column is treated as a four-term polynomial over $\mathrm{GF}(2^8)$ and multiplied modulo $x^4 + 1$ with the polynomial

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

This can be written as a matrix multiplication

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix},$$

where $s_{i,c}$ (resp., $s'_{i,c}$) is the i-the byte of the column c of the old (resp., new) state.

To decrypt, the recipient first derives all round keys from the secret key. Next, he undoes each encryption round by using the round keys in reverse order. Each of the transformations SubBytes, ShiftRows and MixColumns is a bijection and can be easily inverted. To undo the AddRoundKey transformation, we just need to XOR the state and the round key.

Attacks on block ciphers

As mentioned above, a block cipher is considered to be secure if there is no known algorithm that can break it faster than exhaustive search. Some of the most common types of attacks that are used to test the security of block ciphers are differential, linear and related-key attacks.

Differential cryptanalysis exploits the predictability of the propagation of the plaintext difference through the rounds of the block encryption algorithm. An r-round differential is a pair (α, β), where α is the input difference of the block encryption algorithm, and β is the difference of the outputs after r rounds. The probability of an r-round differential (α, β) is the probability that the difference of the outputs of the r-th round will be β, when the input difference is α and the secret key is selected uniformly at random. A possible differential attack works as follows:

1. Find highly probable $(N_r - 1)$-round differential, where N_r is the number of rounds of the block cipher.
2. Select randomly x_1 and submit it for encryption to obtain ciphertext y_1. Compute $x_2 = x_1 + \alpha$ and submit it for encryption to obtain the ciphertext y_2.
3. Find all possible last round keys k_{N_r} such that the difference between $d_{k_{N_r}}(y_1)$ and $d_{k_{N_r}}(y_2)$ is β, where $d_{k_{N_r}}(y)$ is the output of the first round of the decryption algorithm for input y and round key k_{N_r}. Add one to each counter that corresponds to one of the previously computed keys.
4. Repeat previous two steps until one or more last round keys are counted significantly more than the others. Check these keys if they are the right keys.

Linear cryptanalysis exploits the existence of unbalanced linear relations between the input and output bits of the non-linear transformations, where by unbalanced we mean satisfied with probability different than 1/2. A linear trail is a chain of linear expressions involving round input bits, round output bits and round key bits with the property that the output bits involved in the linear expression for round $r - 1$ are the input bits involved in the linear expression for round r. If we combine all linear expressions of the trail, then we will get a linear expression between the cipher input bits, the cipher output bits and the key bits. If the attacker can find such linear expression whose probability is different than 1/2, then he can deduce information about the key and reduce the exhaustive search.

Related-key cryptanalysis makes an additional assumption that the attacker knows a relation between two (or more) different keys that are in use. In chosen plaintext (resp., ciphertext) attacks, the attacker can select a plaintext (resp., ciphertext) of his choice and submit it for encryption (resp., decryption). In a related-key attack, the attacker can submit such queries to encryption/decryption oracles that use different keys. The attacker does not know the keys, but he knows a relation between the keys (e.g., one is derived by rotating the other one by three bytes left).

1.1.2 Pseudorandom functions

An ensemble of functions is pseudorandom if there is no efficient distinguisher that can tell apart the functions of the ensemble from truly random functions even if it can get the values of the functions at arguments of its choice. The notions of function ensembles and pseudorandom function ensembles are formalized by the following two definitions.

Definition 1.1 *Let $l : \mathbf{N} \to \mathbf{N}$. An* l-bit function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbf{N}}$ of random variables such that the random variable F_n assumes values in the set of functions mapping $l(n)$-bit-long strings to $l(n)$-bit-long strings. The* uniform l-bit function ensemble, *denoted $H = \{H_n\}_{n\in\mathbf{N}}$, has H_n uniformly distributed over the set of all functions mapping $l(n)$-bit-long strings to $l(n)$-bit-long strings.*

Definition 1.2 *An l-bit function ensemble $F = \{F_n\}_{n\in\mathbf{N}}$ is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$|\Pr[M^{F_n}(1^n) = 1] - \Pr[M^{H_n}(1^n) = 1]| < \frac{1}{p(n)}$$

where $H = \{H_n\}_{n\in\mathbf{N}}$ is the uniform l-bit function ensemble.

Usually, we are interested in ensembles that can be efficiently computed. The efficiently computable function ensembles are defined as follows.

Definition 1.3 *An l-bit function ensemble $F = \{F_n\}_{n\in\mathbf{N}}$ is called* efficiently computable *if the following two conditions hold:*

1. Efficient indexing: *There exists a polynomial-time algorithm I and a mapping from strings to functions ϕ such that $\phi(I(1^n))$ and F_n are identically distributed. We denote by f_i the functions assigned to the string i (i.e., $f_i = \phi(i)$).*

2. Efficient evaluation: *There exist a polynomial-time algorithm V such that $V(i,x) = f_i(x)$ for every i in the range of $I(1^n)$ and $x \in \{0,1\}^{l(n)}$.*

The existence of pseudorandom functions is closely related to the existence of pseudorandom generators and one-way functions.

Theorem 1.1 *Pseudorandom functions exist if and only if pseudorandom generators exist.*

The case when the members of the function ensemble are permutations is of special interest in cryptography. The permutation ensembles are defined as follows.

Definition 1.4 *A* permutation ensemble *is a sequence $P = \{P_n\}_{n\in\mathbf{N}}$ of random variables such that the random variable P_n assumes values in the set of permutations mapping n-bit-long strings to n-bit-long strings. The* uniform permutation ensemble*, denoted by $K = \{K_n\}_{n\in\mathbf{N}}$, has K_n uniformly distributed over the set of all permutations mapping n-bit-long strings to n-bit-long strings.*

The definition of pseudorandom permutation ensembles is straightforward.

Definition 1.5 *A permutation ensemble $P = \{P_n\}_{n\in\mathbf{N}}$ is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$|\Pr[M^{P_n}(1^n) = 1] - \Pr[M^{K_n}(1^n) = 1]| < \frac{1}{p(n)}$$

where $K = \{K_n\}_{n\in\mathbf{N}}$ is the uniform permutation ensemble.

For cryptographic purposes, we are interested in pseudorandom permutation that can be efficiently computed and inverted. The efficiently computable and invertible permutation ensembles are defined as:

Definition 1.6 *An permutation ensemble $P = \{P_n\}_{n\in\mathbf{N}}$ is called* efficiently computable and invertible *if the following three conditions hold:*

1. Efficient indexing: *There exists a probabilistic polynomial-time algorithm I and a mapping from strings to permutations ϕ such that $\phi(I(1^n))$ and P_n are identically distributed.*

2. Efficient evaluation: *There exists a probabilistic polynomial-time algorithm V such that $V(i,x) = f_i(x)$, where $f_i = \phi(i)$.*

3. Efficient inversion: *There exists a probabilistic polynomial-time algorithm N such that* $N(i, x) = f_i^{-1}(x)$.

It is obvious that pseudorandom functions and pseudorandom ensembles are closely related. Namely, the uniform permutation ensemble constitutes a pseudorandom function ensemble. Also, given a uniform function ensemble, one can construct a pseudorandom permutation ensemble.

1.1.3 Unconditionally secure message authentication

The authentication model

In the authentication model that is usually analyzed, there are three participants: a *transmitter*, a *receiver* and an *adversary*. The transmitter wants to communicate the *source state* (or *plaintext*) to the receiver using a public communication channel. First, the plaintext is encrypted using some *encoding rule* (or *key*). The derived *message* (or *ciphertext*) is sent through the public channel. The transmitter has a key source from which he obtains a key. Prior to any message being sent, the key is communicated to the receiver through a secure channel. The receiver uses the key to verify the validity of the received message. Although the model is communication-based, it is much broader. Namely, the information to be authenticated may indeed be a message in a communication channel, but it can also be stored data.

We use the following notation: $\mathcal{S}$ is the set of plaintexts (source states), $\mathcal{M}$ is the set of all possible messages derived by applying each encoding rule to the source states, and $\mathcal{E}$ is the set of encoding rules. We will use X, Y and Z to denote random variables that take values from $\mathcal{S}, \mathcal{M}$ and $\mathcal{E}$ correspondingly.

An *authentication code* (or A-code) can be represented by an $|\mathcal{E}| \times |\mathcal{M}|$ *encoding matrix* A. The rows of the matrix are indexed by the encoding rules $e_z \in \mathcal{E}$ and the columns of the matrix are indexed by the messages $y \in \mathcal{M}$. The entries of the matrix are either empty or contain a source state. If the message $y \in \mathcal{M}$ is valid under the encoding rule e_z, then there is a plaintext $x \in \mathcal{S}$ such that $y = e_z(x)$. In that case, the entry $A[e_z, y]$ in the encoding matrix should be x. Furthermore, the transmitter should be able to send any plaintext to the receiver regardless of the encoding rule in use. Therefore, for each encoding rule e_z and each source state x, there is at least one message $y \in \mathcal{M}$ such that $y = e_z(x)$. If there is exactly one such message regardless of the plaintext and the encoding rule, then the code is deterministic. Otherwise, the code is randomized. If given a valid message, the adversary can uniquely determine the corresponding source state, then the code is without secrecy. Otherwise, the authentication code is with secrecy.

Table 1.2: An example of a binary A-code with secrecy

	000	011	101
e_1	0	1	
e_2	1		0
e_3		0	1

It is assumed that the secret key will be used only once (although this can be relaxed), and that the adversary has unlimited computational power. Since the key is used only once, there are only two possible types of threats: *impersonation* and *substitution*. In an impersonation attack, the adversary, based only on his knowledge of the authentication scheme, can send a fraudulent message to the receiver when in fact no message has yet been sent by the transmitter. In a substitution attack, the adversary can intercept one valid message and replace it with his fraudulent message. The *probability of successful impersonation*, P_I, is defined as the probability of success when the adversary employs optimum impersonation strategy. The *probability of successful substitution*, P_S, is defined as the probability of success when the adversary employs optimal substitution strategy. Finally, the adversary may be able to select whether to employ an impersonation or a substitution attack (a *deception attack*). The *probability of successful deception*, P_d, is the probability of success when an optimum deception strategy is employed. Assuming that the encoding strategy is fixed and known to all participants, it holds that $P_d = max(P_I, P_S)$.

A toy example of an encoding matrix of a binary authentication code is given in Table 1.2. If the keys are equiprobable, then the probability of successful substitution is $P_S = 1/2$, and the probability of successful impersonation is $P_I = 2/3$.

Deception lower bounds

What is the maximum security that can be achieved by an authentication code? The following theorem provides some answers.

Theorem 1.2 *The following inequalities hold for the probability of successful deception:*

1. $P_d \geq \frac{|\mathcal{S}|}{|\mathcal{M}|}$
2. $P_d \geq 2^{-I(Y,Z)}$
3. $P_d \geq \frac{1}{\sqrt{|\mathcal{E}|}}$

where $I(Y, Z)$ denotes the mutual information between Y and Z.

The first inequality gives a lower bound given the number of source states and possible messages. The second inequality is known as the authentication channel capacity, and it reveals a little surprising fact that the more information the message leaks about the key, the more secure the code can be. The final inequality gives an answer to the question what is the maximum achievable security for a given key size.

Universal classes of hash functions

The notion of universal classes of hash functions was introduced in order to study collections of hash functions with the following property: the hash values of two distinct inputs collide with small probability when the hash function is selected randomly from the collection. Although the original motivation was to provide an input independent algorithm for storage and retrieval in a hash table, one of the applications of universal classes of hash functions is to construct unconditionally secure authentication codes without secrecy.

Let A and B be finite sets with cardinalities $|A| = a$ and $|B| = b$, where $a \geq b$. Given a hash function $h : A \rightarrow B$ and two elements $x, y \in A$, the characteristic function $\delta_h(x, y)$ is defined as

$$\delta_h(x, y) = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

For a finite collection H of hash functions, the number of hash functions in H under which x and y collide is $\delta_H(x, y) = \sum_{h \in H} \delta_h(x, y)$. The probability that x and y will collide for a hash function that is selected uniformly at random will be $\delta_H(x, y)/|H|$. For the universal classes of hash functions, this probability should be small as stated by the following definition.

Definition 1.7 *Let H be a class of hash functions from A to B.*

1. *H is* universal$_2$ *if $\delta_H(x, y) \leq |H|/|B|$ for all $x, y \in A$, $x \neq y$.*

2. *H is* ϵ-almost universal$_2$ *if $\delta_H(x, y) \leq \epsilon|H|$ for all $x, y \in A$, $x \neq y$, where ϵ is a positive real number.*

3. *Let ϵ be a positive real number. H is* ϵ-almost strongly-universal$_2$ *if the following two conditions are satisfied:*

   - *for every $x_1 \in A$ and for every $y_1 \in B$, $|\{h \in H : h(x_1) = y_1\}| = |H|/|B|$*

- *for very* $x_1, x_2 \in A$ $(x_1 \neq x_2)$ *and for every* $y_1, y_2 \in B$,

$$|\{h \in H : h(x_1) = y_1, h(x_2) = y_2\}| \leq \epsilon |H|/|B|.$$

The next result immediately follows from the previous definition.

Theorem 1.3 *If there exists an* ϵ*-almost strongly-universal*$_2$ *class* H *of hash functions from* A *to* B*, then there exists an authentication code for* $|A|$ *source states, having* $|B|$ *authenticators and* $|H|$ *encoding rules, such that* $P_I = 1/|B|$ *and* $P_S \leq \epsilon$.

1.1.4 Signature schemes

Message authentication vs digital signing

Message authentication codes (MACs) and digital signature schemes are two common ways of achieving computationally secure data origin authentication. The data origin authentication mechanisms based on shared secret keys (e.g., MACs) do not allow a distinction to be made between parties sharing the key, and thus (as opposed to digital signatures) do not provide non-repudiation of data origin, i.e., either party can equally originate a message using a shared key.

Specifically, a *scheme for unforgeable signatures* should satisfy the following:

- Each user can efficiently produce his own signature on documents of his choice.
- Every user knowing the appropriate public key can efficiently verify that a given string is a signature of another user on a particular document.
- It is infeasible to produce signatures of other users on documents they haven't signed.

The schemes for message authentication should satisfy the following:

- Each of the communicating parties can efficiently generate an authentication tag on a message of his choice.
- Each of the communicating parties can efficiently verify whether a given string is a valid authentication tag for a given message.
- It is infeasible for an external adversary (i.e., a party different from the communicating parties) to produce a valid authentication tag for messages different from those sent by the communicating parties.

Definition

From the previous discussion, it follows that the unforgeable digital signature schemes and the unforgeable message authentication schemes differ only in the third requirement. In both cases, the authenticity of the data is established by a procedure consisting of two main algorithms:

- a *signing algorithm* that is used by the sender to produce signatures on messages of his choice, and
- a *verification algorithm* that is used to verify the authenticity of the message by checking the validity of the provided signature.

Since the third requirement is a security requirement, the difference between message authentication and digital signing will be reflected only in the security definition, and a unique definition for the basic mechanism of message authentication and signature schemes can be used. We will refer to this mechanism as signature scheme in both cases.

Definition 1.8 *A* signature scheme *is a triple* (G, S, V) *of probabilistic polynomial-time algorithms satisfying the following two conditions:*

1. *On input* 1^n*, algorithm* G *(called the* key generator*) outputs a pair of bit strings.*
2. *For every pair* (s, v) *in the range of* $G(1^n)$*, and for very* $\alpha \in \{0,1\}^*$*, algorithms* S *(*signing*) and* V *(*verification*) satisfy*

$$\Pr[V(v, \alpha, S(s, \alpha)) = 1] = 1$$

where the probability is taken over the internal coin tosses of S *and* V*.*

Each pair (s, v) *constitutes a pair of corresponding* signing/verification keys*. The string* $S(s, \alpha)$ *is called a* signature to the document α *using the signing key* s*.*

The only requirement imposed by the previous definition is that the signatures produced by the signing algorithm must be accepted as valid by the verification algorithm. The definition says nothing about security of the scheme and so even insecure algorithms may satisfy it.

Attacks and security

In this subsection, we will consider a very strong definition of security. Namely, the adversary is allowed a chosen message attack, and the attack is considered to be successful if the adversary

manages to produce a valid signature to any document for which it has not asked for signature during the attack. This leads to the following informal formulation:

- The adversary is allowed chosen message attack, that is the attacker can query for signatures of messages of his own choice. We distinguish two cases:

  1. The private key case: The attacker is given 1^n as input, and the signatures are produced relative to s, where $(s, v) \leftarrow G(1^n)$.
  2. The public key case: The attacker is given v as input, and the signatures are produced relative to s, where $(s, v) \leftarrow G(1^n)$.

- The attack is considered successful if the adversary outputs a valid signature to a string for which it has not requested a signature during the attack.

- A signature scheme is considered secure (or unforgeable) if every feasible attack succeeds with at most negligible probability.

Formally, the adversary in a chosen message attack is modeled using a probabilistic oracle machine that is given access to a keyed signing oracle. Depending on the type of the scheme (public key or private key), the adversary is given the verification key v as input or not.

Definition 1.9 *For a probabilistic oracle machine M, we denote by $Q_M^O(x)$ the set of queries made by M on input x and access to oracle O. As usual, $M^O(x)$ denotes the output of the corresponding computation. We stress that $Q_M^O(x)$ and $M^O(x)$ are dependent random variables that describe two different aspects of the same probabilistic computation.*

1. The private key case: *A private key signature scheme is secure if for every polynomial-time probabilistic oracle machine M, every positive polynomial p and all sufficiently large n, it holds that*

$$\Pr[V(v, \alpha, \beta) = 1 \text{ and } \alpha \notin Q_M^{S(s,\cdot)}(1^n)] < \frac{1}{p(n)}$$

   where $(s, v) \leftarrow G(1^n)$ and $(\alpha, \beta) \leftarrow M^{S(s,\cdot)}(1^n)$. The probability is taken over the coin tosses of G,S and V as well as over the coin tosses of machine M.

2. The public key case: *A public key signature scheme is secure if for every polynomial-time probabilistic oracle machine M, every positive polynomial p and all sufficiently large n, it holds that*

$$\Pr[V(v, \alpha, \beta) = 1 \text{ and } \alpha \notin Q_M^{S(s,\cdot)}(v)] < \frac{1}{p(n)}$$

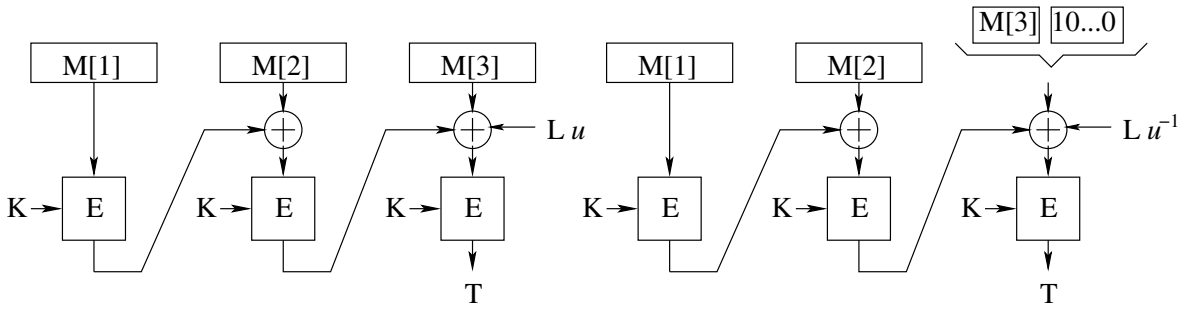

Figure 1.3: OMAC ($l = 3$)

where $(s, v) \leftarrow G(1^n)$ *and* $(\alpha, \beta) \leftarrow M^{S(s,\cdot)}(v)$. *The probability is taken over the coin tosses of* G,S *and* V *as well as over the coin tosses of machine* M.

Construction of secure signature schemes

There are a number of private key and public key signature schemes that are proven secure assuming the underlying cryptographic primitives are secure.

Pseudorandom functions can be used to construct a secure private key signature scheme (MAC scheme). The key is used to select a function from the ensemble. The input of the function is the message and the output is the authentication tag. The receiver also computes the authentication tag on receipt of the message and accepts the message as authentic if the provided tag is equal to the computed one. Block encryption algorithms are often used as underlying cryptographic primitive to construct a MAC scheme. The security of the MAC scheme is proved by proving that the family of functions defined by the MAC scheme is pseudorandom if the family of permutations defined by the underlying block cipher is pseudorandom.

Usually, the hash-and-sign paradigm is used to construct secure public key signature schemes (digital signatures). Namely, the message to be signed is first hashed using some hash function, and then the result is signed using a public key scheme based on a trapdoor one-way function.

One-key CBC MAC

OMAC is a message authentication scheme. It operates in a CBC-like fashion as illustrated in Fig 1.3.

If the message length $|M|$ is a multiple of the block size n, then the message is divided into blocks $M[1], M[2], \ldots, M[l]$, each one of length n. First, the block $M[1]$ is encrypted using some block

cipher with secret key K. The result $C[1]$ is XOR-ed with the second block $M[2]$ and encrypted to get $C[2]$, and so on. Before being encrypted, the last block $M[l]$ is XOR-ed not only with $C[l-1]$, but with $L \cdot u$ also, where u is some constant in $\mathrm{GF}(2^n)$ and $L = E_K(0^n)$ is the encryption of zero. The derived value $C[l]$ is used as an authentication tag. The sender will send both the message M and the authentication tag T. The receiver will recompute the tag from the message and if it is equal to the one that was received he will accept the message as valid.

If the message length $|M|$ is not a multiple of the block size, then a 10^i padding is appended, where $i = n - 1 - |M| \bmod n$. The tag computation proceeds as in the previous case with only one difference. Instead of XOR-ing with $L \cdot u$, the last block is XOR-ed with $L \cdot u^{-1}$.

The scheme is secure if the underlying block cipher is secure as stated informally by the following theorem.

Theorem 1.4 *OMAC is a pseudorandom function if the underlying block cipher is a pseudorandom permutation.*

Digital Signature Algorithm

Digital Signature Algorithm was first proposed in 1991. On May 19, 1994, it was published in the Federal Register, and on December 1, 1994, it was adopted as a standard. It is a modification of the well known ElGamal digital signature scheme. The signing and verification algorithms of the scheme are described below.

Let p be a prime whose length is a multiple of 64 between 512 and 1024. Let $q|p-1$ be a 160-bit prime, and let $\alpha \in \mathbb{Z}_p^*$ be a qth root of 1 modulo p. The set of messages[1] is $\mathcal{P} = \mathbb{Z}_q^*$, the set of digital signatures is $\mathcal{A} = \mathbb{Z}_q \times \mathbb{Z}_q$, and the set of keys is

$$\mathcal{K} = \{(p, q, \alpha, a, \beta) : \beta = \alpha^a \bmod p\},$$

where p,q,α and β are public, and a is secret.

Given a $K \in \mathcal{K}$ and a message $x \in \mathcal{P}$, the signing algorithm selects randomly a secret random number $k \in \mathbb{Z}_q - \{0\}$, and computes the signature as

$$sig_K(x, k) = (\gamma, \delta)$$

where

$$\gamma = (\alpha^k \bmod p) \bmod q$$

[1]Actually, $\mathcal{P}$ is the set of possible hash values of the messages. To simplify, we assume that all messages belong to $\mathcal{P}$, and there is no need of hash functions.

and

$$\delta = (x + a\gamma)k^{-1} \bmod q.$$

Given a message x and a signature (γ, δ), the verifier computes the following values:

$$e_1 = x\delta^{-1} \bmod q,$$

$$e_2 = \gamma\delta^{-1} \bmod q.$$

The signature is considered valid if and only if

$$(\alpha^{e_1}\beta^{e_2} \bmod p) \bmod q = \gamma.$$

1.1.5 Stream Authentication Schemes

Streams are bit sequences of potentially unknown length that a sender sends to one or more recipients at some rate negotiated between the sender and the receivers. Examples include audio, video, data feeds, aplets, etc. Usually, the streams are divided into chunks that are sent in different packets. The receiver processes the chunks as they arrive. Hence, the most convenient way to achieve stream authentication is by including some authentication information (tag or signature) in the packets so that the authenticity of the chunk can be verified immediately.

Gennaro-Rohatgi

Gennaro and Rohatgi have proposed a stream signing scheme based on a chain of one-time signatures[2]. Namely, a standard digital signature algorithm is used to sign the first chunk which contains a public key for a one-time signature scheme. The corresponding secret key is used to sign the second chunk together with a one-time public key that will be used in the third step, and so on. The scheme solves the scaling problem because the authentication information is independent of the number of recipients, and it is also more efficient than using standard digital signature since one-time signatures are more efficient than standard ones.

Let $M_1, M_2, M_3, \ldots$ be the sequence of the stream chunks. Given a regular signature scheme (G, S, V), a one-time signature scheme (g, s, v) and a one-way function H, the signing algorithm of the Gennaro-Rohatgi scheme computes a sequence $M'_0, M'_1, M'_2, \ldots$, where

$$\begin{array}{rcll} M'_0 & = & \langle pk_0, S(SK, pk_0)\rangle & \\ M'_i & = & \langle M_i, pk_i, s(sk_{i-1}, H(M_i, pk_i))\rangle & \text{for } i > 0 \end{array}^{y}$$

[2] One-time signature schemes are signature schemes where the secret key is used only once

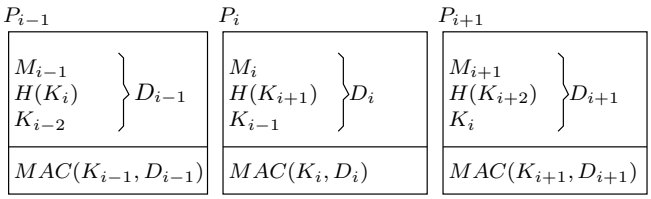

Figure 1.4: The basic stream authentication scheme

The authenticity of the chunks is verified as follows. On receiving $M'_0 = \langle pk_0, A_0 \rangle$, the receiver checks that

$$V(PK, pk_0, A_0) = 1$$

and then on receiving $M'_i = \langle M_i, pk_i, A_i \rangle$ he checks that

$$V(pk_{i-1}, H(M_i, pk_i), A_i) = 1.$$

Whenever one of these checks fails, the receiver stops processing the stream.

Timed Efficient Stream Loss-tolerant Authentication

The Timed Efficient Stream Loss-tolerant Authentication scheme is a multicast stream authentication scheme proposed. Here, we briefly describe the mechanisms employed in TESLA to achieve loss-tolerance, fast transfer rates and dynamic packet rates.

The security of TESLA is based on the paradigm depicted in Figure 1.4. To authenticate the packet P_i of the stream, the sender first commits to the key value K_i by sending $H(K_i)$ in the packet P_{i-1}. The key K_i is only known to the sender, and it is used to compute a MAC on the packet P_i. After all recipients have received the packet P_i, the sender discloses the key value K_i in the packet P_{i+1}. The recipient verifies whether the received key value corresponds to the commitment and whether the MAC of the packet P_i computed using the received key value corresponds to the received MAC value. If both verifications are successful, the packet P_i is accepted as authentic. Note that P_i contains the commitment to the next key value K_{i+1}. To bootstrap the scheme, the first packet is signed using a digital signature scheme (e.g., RSA). If the packet P_{i-1} is lost, then the authenticity of the packet P_i and all subsequent packets cannot be verified since the commitment to the key K_i is lost. Similarly, if the packet P_{i+1} is lost, the authenticity of the packet P_i and all subsequent packets cannot be verified since the key K_i is lost.

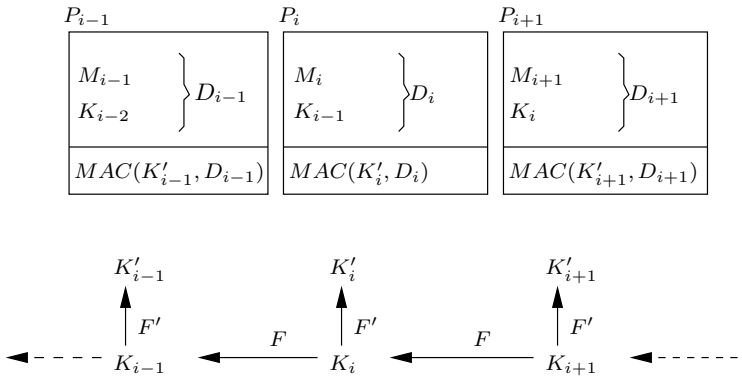

Figure 1.5: TESLA Scheme II: Tolerating packet loss

Perrig at al. [88] proposed a solution to the above problem by generating the sequence of keys K_i using iterative application of a pseudo-random function to some initial value as illustrated in Figure 1.5. Let us denote v consecutive applications of the pseudo-random function F as $F^v(x) = F^{v-1}(F(x))$, and let $F^0(x) = x$. The sender has to pick randomly some initial key value K_n and to pre-compute n key values $K_0, \ldots, K_{n-1}$, where $K_i = F_{n-i}(K_n), i = 0, \ldots, n$. The sequence of key values is called a *key chain*. The key K'_i, which is used to authenticate the packet P_i, is derived from the corresponding key value K_i by applying the function F'. Since F is easy to compute and hard to invert, given K_i the attacker cannot compute any K_j for $j > i$. However, the recipient can compute any key value K_j from the received key value K_i, where $j < i$. Therefore, if the recipient has received a packet P_i, any subsequently received packet P_j $(j > i)$ will allow computation of $K'_i = F'(K_i)$ and verification of the authenticity of the packet P_i.

The authors suggest the function F to be implemented as $F(K_i) = f_{K_i}(0)$, where f is a target collision resistant pseudorandom function (i.e., given $f_{K_i}(0)$ it is hard to find a key K such that $f_K(0) = f_{K_i}(0)$). There are no requirements imposed on the function F' in the original description of TESLA [88]. However, RFC4082 requires $F'(K_i)$ to be computed as $F'(K_i) = f'_{K_i}(1)$, where f' is a pseudorandom function.

The security of the scheme is based on the assumption that the receiver can decide whether a given packet arrived safely (i.e., before the corresponding key disclosure packet was sent by the sender). The unsafe packets are dropped. This condition severely limits the transmission rate since P_{i+1} can only be sent after every receiver has received P_i. Perrig et al [88] solve this problem by disclosing the key K_i of the data packet P_i in a later packet P_{i+d}, instead of in the next packet.

Another assumption made in the scheme depicted in Figure 1.5 is that the packet schedule is fixed or predictable, with each recipient being able to estimate the sending time of each packet. This severely restricts the flexibility of the senders. The proposed solution to this problem of dynamic packet rates is to pick the MAC key and the disclosed key in each packet only on a time interval basis. Namely, all packets sent in an interval i are authenticated using a key K_i and disclose the key K_{i-d}. This final version (TESLA Scheme IV) is the one adopted as an Internet standard.

The following theorem about the security of TESLA was claimed by the authors.

Theorem 1.5 *Assume that the PRF, the MAC and the signing schemes in use are secure, and that the PRF has Target Collision Resistance property. Then, TESLA (Scheme IV) is a secure stream authentication scheme.*

To avoid complexity, the authors provided a proof only for the case when the MAC and the PRF are realized by the same function family. In their implementation, this family is the family defined by HMAC when used in conjunction with MD5. In Chapter 4, we will show that the theorem does not hold in the general case when the PRF and the MAC can be realized by different function families.

1.2 Organization of the dissertation and our contributions

The dissertation is organized as follows.

In Chapter 2, we address several fundamental issues concerning erasure-tolerant authentication codes. We analyze a setting where both unconditionally secure information authentication and loss-tolerance are achieved at the same time via erasure-tolerant authentication codes (or η-codes). We adapt some lower bounds on the probability of successful deception derived for the traditional authentication model to the setting that we consider here. We also analyze the distance properties of the η-codes and the security properties of the η-codes constructed by means of concatenation and composition. One interesting class of η-codes is the class of η-codes with minimal probabilities of successful impersonation and substitution. We show that all members of this class can be represented as a composition of an authentication code with minimal impersonation and substitution probabilities and an erasure-resilient code. Finally, we present some examples of η-code constructions.

The case of unicast stream authentication is discussed in Chapter 3. We present several efficient schemes and also a family of schemes for stream authentication in a unicast setting. Since many

authentication schemes have been broken, we prove our solutions.

In Chapter 4, we study the problem of multicast stream authentication. We analyze the security of the Timed Efficient Stream Loss-tolerant Authentication scheme (RFC4082), and show that the assumptions about the security of the components of TESLA are not sufficient. In particular, we present examples of insecure TESLA implementations constructed from secure components. We also provide sufficient assumptions and propose implementations based on related-key secure block ciphers and erasure-tolerant authentication codes.

Chapter 5 investigates the resistance of AES-192 to related-key differential cryptanalysis. We were able to break six rounds using related-key differentials and related-key truncated differentials. We also present attacks on seven and eight rounds using impossible related-key differentials. In the case of differential cryptanalysis, if the iterated cipher is Markov cipher and the round keys are independent, then the sequence of differences at each round output forms a Markov chain and the cipher becomes resistant to differential cryptanalysis after sufficiently many rounds. However, this is not sufficient in the case of related-key differentials. We show that if in addition the Markov cipher has $\mathcal{K} - f$ round function and the hypothesis of stochastic equivalence for related keys holds, then the iterated cipher is resistant to related-key differential attacks after sufficiently many rounds.

1.3 Further reading

The concepts of one-way functions and trapdoor permutations were introduced by Diffie and Hellman [35]. The definition of weak one-way function is due to Yao [117]. The relations between weakly one-way functions and strongly one-way functions have been studied by Goldreich [48]. Rivest, Shamir and Adleman [97] introduced the RSA function. Other suggestions for one-way functions can be found in [7, 46, 55].

The notion of computationally indistinguishable ensembles originates from the paper of Blum and Micali [20]. The notion of computational indistinguishability in the general setting and the notion of pseudorandomness are due to Yao [117]. Blum and Micali [20] were the first to define pseudorandom number generators. However, they defined the pseudorandom generators as producing sequences that are unpredictable. Yao [117] defined pseudorandom generators as producing sequences that are indistinguishable from uniform sequence and showed that the two definitions are equivalent. The relations between one-way functions and pseudorandom generators has been studied in [117, 73, 46, 54]. Some constructions of pseudorandom generators can be found in [20, 19, 2, 112].

Goldreich, Goldwasser and Micali [45] introduced and studied pseudorandom functions. Applications of pseudorandom functions are given in [49]. Pseudorandom permutations were introduced by Luby and Rackoff [74].

AES [41] is derived from Rijndael [30], which was proposed by Daemen and Rijmen. The techniques employed for the design of Rijndael were presented in [31]. Differential cryptanalysis [11, 12] and related-key cryptanalysis [13] were introduced by Biham et al. The linear cryptanalysis [77] is due to Matsui. Some other applications and generalizations of these attacks can be found in [70, 71, 67, 65].

Authentication codes have been extensively studied in the past. Some of the fundamental results include the following. Simmons [102, 103] developed the theory of unconditional authentication and derived some lower bounds on the deception probability. Stinson [106, 107, 108] (see also [62]) studied the properties of authentication codes that have minimum possible deception probabilities and minimum number of encoding rules. He characterized the existence of authentication codes that have the aforementioned desirable properties in terms of existence of orthogonal arrays and balanced incomplete block designs (BIBD). The problem of unconditionally secure multicast message authentication has been studied by Desmedt et al [33]. Carter and Wegman [25, 115] introduced the notion of universal classes of hash function and proposed their use in authentication. The use of universal hashing to construct unconditionally secure authentication codes without secrecy has also been studied by Stinson [109] and by Bierbrauer et al [10]. Afanassiev et al [1] proposed an efficient procedure for polynomial evaluation that can be used to authenticate messages in about 7-13 instructions per word.

The notion of digital signatures was introduced by Diffie and Hellman [35], and a first practical realization appeared in [97]. The security notions for digital signature schemes are discussed in [53]. The DSA algorithm is described in [39]. Other digital signature schemes and security analysis can be found in [18, 32, 52, 72, 79, 80, 94, 101]. CBC-MAC [38] is one of the most popular MAC algorithms. Bellare at al [5] introduced the notion of exact security and provided a security proof for CBC-MAC when the length of the input is fixed. There is a simple attack that breaks the scheme when the length of the input is not fixed. Subsequent analysis and solutions to this problem were presented in [8, 92, 16, 61], and culminated with the OMAC scheme [57], which was proposed by Iwata and Kurosawa. Other MAC schemes, some of them having desirable properties such as parallelizability and incrementality, can be found in [4, 6, 17, 42].

The Gennaro-Rohatgi scheme [43] solves the scaling problem because the authentication

information is independent of the number of recipients, and it is also more efficient than using standard digital signature since one-time signatures are more efficient than standard ones. The major disadvantages of the scheme are the large length of the authentication information and its non-resilience to packet loss. The first disadvantage was solved by the Guy Fawkes protocol [3] (see also Cheung [28]) through the use of hash function to bind a sequence of events to an initial value and to guarantee the sequence integrity. However, both schemes suffer from the second disadvantage. Namely, they will collapse in the case of packet loss. Perrig et al. [88] and Bergadano et al. [9] independently proposed a solution to the problem of multicast stream authentication in networks with high packet loss rates. A somewhat different approach to stream signing is presented in [83, 116, 118].

CHAPTER 2

Erasure-tolerant Information Authentication

The traditional authentication model assumes that the data loss on the communication channel between the sender and the receiver is handled by mechanisms that should be studied separately. Here, we consider a more general setting where both unconditionally secure information authentication and loss-resilience are achieved at the same time via erasure-tolerant authentication codes (or η-codes)[1].

2.1 The setting

The authentication model that we are going to analyze is based on the model described in [102, 103, 76]. As it is case in the usual model, there are three participants: a *transmitter*, a *receiver* and an *adversary*. The transmitter wants to communicate the *source state* (or *plaintext* [2]) to the receiver using a public communication channel. We assume that all plaintexts are strings of length k whose letters are from some q-set Q. First, the plaintext is encrypted using some *encoding rule* (or *key*) into a q-ary string of length n. The derived *message* (or *ciphertext*) is sent through the public channel. The transmitter has a key source from which he obtains a key. Prior to any message being sent, the key is communicated to the receiver through a secure channel. The receiver uses the key to verify the validity of the received message. If at most t ($t < n$) letters are missing from the original intact valid message and the position of the missing letters within the message is known, then the received message is still considered valid. In this case, the receiver accepts a plaintext that is derived from the original plaintext by erasing at most r ($r < k$) letters. When r is zero (i.e., we can recover and verify the authenticity of the complete plaintext), we say that

[1] Some of the results in this chapter were presented in [59].

[2] In general, η-codes can provide secrecy (privacy). Hence, we adhere to the terminology used by Simmons and use the terms plaintext (or source state), ciphertext (or message) and encryption. This can be slightly confusing, since most of the current works on MAC schemes use the term message for the source state, the term message authentication code only for codes without secrecy, and the term authenticated encryption for schemes that provide both authenticity and privacy.

the code is *full-recovery*. If the received message is not derived from some intact valid message by erasing at most t letters, then the receiver does not accept the plaintext.

We denote by $\mathcal{S}$ the set of plaintexts (source states). Let $\mathcal{M}_0$ be the set of all possible messages derived by applying each encoding rule (key) to the source states and let $\mathcal{M}_i$ $(0 < i \leq t)$ be the set of all possible messages derived by erasing i letters from the messages in $\mathcal{M}_0$. The set of all possible messages is $\mathcal{M} = \bigcup_{i=0}^{t} \mathcal{M}_i$. Finally, we will use $\mathcal{E}$ to denote the set of encoding rules, and $X, Y_0, \ldots, Y_t, Y$ and Z to denote random variables that take values from $\mathcal{S}, \mathcal{M}_0, \ldots \mathcal{M}_t, \mathcal{M}$ and $\mathcal{E}$ correspondingly. Note that the probability distribution of Y_0 is uniquely determined by the probability distributions of X, Z and the randomness used by the code. However, there are infinitely many possible probability distributions for each of the random variables $Y_1, \ldots, Y_t$ depending on how we erase the letters of the messages.

An η-code can be represented by an $|\mathcal{E}| \times |\mathcal{M}|$ *encoding matrix* A. The rows of the matrix are indexed by the encoding rules $e_z \in \mathcal{E}$ and the columns of the matrix are indexed by the messages $y \in \mathcal{M}$. The entries of the matrix are either empty or contain a string derived from a source state by erasing at most r (possibly 0) letters. We use the characteristic functions $\chi(y, z)$ to indicate whether an entry is empty or not. In particular, the function $\chi(y, z)$ is one if $A[e_z, y]$ is not empty (i.e., $y \in \mathcal{M}$ is a valid message when the key in use is z), and $\chi(y, z)$ is zero if $A[e_z, y]$ is empty (i.e., $y \in \mathcal{M}$ is not a valid message when the key in use is z). We will also use the characteristic function $\phi(y_1, y_2, z)$, which is one if both y_1 and y_2 are valid when encoding rule e_z is used, and zero otherwise.

The following restrictions are imposed on the encoding matrix in order to capture the important aspects of the setting that we have discussed above. Let the message $y \in \mathcal{M}_0$ be valid under the encoding rule e_z. Then, there is a plaintext $x \in \mathcal{S}$ such that $y = e_z(x)$. In that case, the entry $A[e_z, y]$ in the encoding matrix should be x. Furthermore, the transmitter should be able to send any plaintext to the receiver regardless of the encoding rule in use. Therefore, for each encoding rule e_z and each source state x, there is at least one message $y \in \mathcal{M}_0$ such that $y = e_z(x)$. If there is exactly one such message regardless of the plaintext and the encoding rule, then the code is deterministic. Otherwise, the code is randomized. If the message $y \in \mathcal{M}_0$ is valid under the encoding rule e_z, then any message y' derived from y by erasing at most t letters is also valid. The entry $A[e_z, y']$ should be derivable from the entry $A[e_z, y]$ by erasing at most r letters. Note that if we discard from the encoding matrix each column that does not correspond to an element in $\mathcal{M}_0$, then the resulting encoding matrix defines an authentication code. We will refer to this code as

Table 2.1: An example of a binary η-code with secrecy

	000	00*	0*0	*00	011	01*	0*1	*11	101	10*	1*1	*01
e_1	0	0	0	0	1	1	1	1				
e_2	1	1	1	1					0	0	0	0
e_3					0	0	0	0	1	1	1	1

the *underlying authentication code*.

We assume that the secret key will be used only once (although this can be relaxed), and the probabilities of successful impersonation, substitution and deception are defined as in Section 1.1.3.

A simple example of an encoding matrix for a binary η-code is given in Table 2.1. We use * to denote the missing letters. If we discard the columns that correspond to the incomplete messages, then we will get an authentication code. This underlying authentication code is same as the one shown in Table 1.2. Assuming that the keys are equiprobable, the code has the following parameters: $k = 1, n = 3, t = 1, r = 0, \mathcal{S} = \{0, 1\}, \mathcal{M}_0 = \{000, 011, 101\}, \mathcal{E} = \{e_1, e_2, e_3\}, P_S = 1/2, P_d = P_I = 2/3$. Note that the code is full recovery ($r = 0$), and that it is with secrecy since the adversary cannot determine the plaintext given a ciphertext. For example, assume that the adversary has intercepted the message 000. Since he doesn't know the key that is in use, he can not tell whether the plaintext is 0 or 1.

2.2 Lower bounds on the deception probability

In this section, we are going to address the question of how much security we can achieve given some parameters of the η-code. The following theorem gives some lower bounds on the probability of successful deception for the model described in Section2.1.

Theorem 2.1 (Lower Bounds) *The following inequalities hold for the probability of successful deception:*

1. $P_d \geq \dfrac{\min\limits_{e_z \in \mathcal{E}} |\mathcal{M}(e_z)|}{|\mathcal{M}|}$
2. $P_d \geq \dfrac{|\mathcal{S}|}{|\mathcal{M}_0|}$
3. $P_d \geq 2^{-\inf_{Y_t} I(Y_t, Z)} \geq \ldots \geq 2^{-\inf_{Y_1} I(Y_1, Z)} \geq 2^{-I(Y_0, Z)}$
4. $P_d \geq \dfrac{2^{\frac{1}{2}(I(Y_0,Z) - \inf_{Y_t} I(Y_t,Z))}}{\sqrt{|\mathcal{E}|}} \geq \dfrac{1}{\sqrt{|\mathcal{E}|}}$

where $|\mathcal{M}(e_z)|$ *is the number of messages in* $\mathcal{M}$ *that are valid when the key* z *is used, and* $\inf_{Y_i} I(Y_i, Z)$ *denotes the infimum of the mutual information* $I(Y_i, Z)$ *over all possible probability distributions of* Y_i.

Proof. Assume that the sender and the receiver have already agreed that they are going to use the encoding rule e_z. One possible impersonation strategy is to select uniformly at random a message from $\mathcal{M}$. The attack will succeed if the selected message is one of the messages that are valid under that key z. Since there are $|\mathcal{M}(e_z)|$ messages that are valid under the key z and the fraudulent message was selected uniformly at random, the probability of success given a key z is $|\mathcal{M}(e_z)|/|\mathcal{M}|$. Clearly, the probability of successful deception will be greater or equal than $\frac{\min_{\mathcal{E}} |\mathcal{M}(e_z)|}{|\mathcal{M}|}$, which is our first lower bound.

In Section 2.1, we mentioned that if we discard from the encoding matrix all columns that are not indexed by the messages in $\mathcal{M}_0$, then we will get an authentication code. It is obvious that an attack on the underlying authentication code is also an attack on the η-code. Hence, any lower bound on the probability of successful deception for authentication codes can be translated into a lower bound on the probability of success in a deception attack for η-codes. One such bound is given in Theorem 2.1(2) and it follows from Corollary 1 [102].

The third bound (Theorem 2.1(3)) is an erasure-tolerant analogue to the authentication channel capacity bound [102]. The probability that a particular message $y \in \mathcal{M}$ will be a valid message is given by the following expression:

$$P(y \text{ valid}) = \sum_z \chi(y, z) P_Z(z).$$

The probability of successful impersonation is

$$P_I = \max_y P(y \text{ valid}),$$

that is the best impersonation attack is when the adversary selects the fraudulent message to be the message that will be valid with maximum probability. Assume that y is a message from $\mathcal{M}$ that is valid with maximum probability ($P(y \text{ valid}) = P_I$) and assume that $y \notin \mathcal{M}_t$. Let $\hat{y} \in \mathcal{M}_t$ be a message derived from y by erasing some of its letters. Note that $\hat{y}$ is valid whenever y is valid, or equivalently, $\chi(y, z) = 1$ implies $\chi(\hat{y}, z) = 1$. In that case, we have

$$P_I \geq P(\hat{y} \text{ valid}) = \sum_z \chi(\hat{y}, z) P_Z(z) \geq \sum_z \chi(y, z) P_Z(z) = P(y \text{ valid}) = P_I.$$

Obviously, the probability that message $\hat{y}$ will be a valid message is P_I, and one best impersonation strategy is to choose always $\hat{y}$ as a possible fraudulent message.

Let Y_t be a random variable that takes values from $\mathcal{M}_t$ in a following manner: we randomly discard t letters from a message that is computed by the receiver from a randomly selected plaintext (according to the plaintext probability distribution) by applying a randomly selected encoding rule (according to the key probability distribution). It is clear that

$$\begin{aligned} P_I &= P(\hat{y}\,\text{valid}) = P(\hat{y}\,\text{valid}) \times \sum_y P_{Y_t}(y) = \sum_y P_{Y_t}(y) P(\hat{y}\,\text{valid}) \\ &\geq \sum_y P_{Y_t}(y) P(y\,\text{valid}). \end{aligned}$$

Note that equality holds only when the probabilities of validity are equal for all messages in $\mathcal{M}_t$. By substituting $P(y\,\text{valid})$, we get

$$P_I \geq \sum_{y,z} P_{Y_t}(y) P_Z(z) \chi(y,z).$$

The joint probability $P_{Y_t Z}(y,z)$ is greater than zero if and only if $P_Z(z) > 0$ and $\chi(y,z) = 1$. Therefore, the relation above can be rewritten as

$$P_I \geq E\left[\frac{P_{Y_t}(y) P_Z(z)}{P_{Y_t Z}(y,z)}\right].$$

Using Jensen's inequality[3], we get

$$\begin{aligned} \log P_I &\geq \log E\left[\frac{P_{Y_t}(y) P_Z(z)}{P_{Y_t Z}(y,z)}\right] \geq E\left[\log \frac{P_{Y_t}(y) P_Z(z)}{P_{Y_t Z}(y,z)}\right] \\ &= H(Y_t Z) - H(Y_t) - H(Z) = -I(Y_t, Z). \end{aligned}$$

The lower bound $P_d \geq 2^{-\inf_{Y_t} I(Y_t,Z)}$ trivially follows since the previous inequality holds for any probability distribution of Y_t. Now, we only need to show that $\inf_{Y_i} I(Y_i, Z) \leq \inf_{Y_{i-1}} I(Y_{i-1}, Z)$. Given a random variable Y_{i-1} that takes values from $\mathcal{M}_{i-1}$, let us construct a random variable Y_i that takes values from $\mathcal{M}_i$ as follows. If $y_{i-1} \in \mathcal{M}_{i-1}$ is the message that the receiver is supposed to get, we erase the first non-erased letter in y_{i-1} to get y_i. It is obvious that $I(Y_i, Z) \leq I(Y_{i-1})$

[3] If $f(x)$ is a convex function on an interval (a,b), $x_1, x_2, \ldots, x_n$ are real numbers $a < x_i < b$, and $w_1, w_2, \ldots, w_n$ are positive numbers with $\sum w_i = 1$, then

$$f\left[\sum_{i=1}^n w_i x_i\right] \leq \sum_{i=1}^n w_i f(x_i).$$

since anything that we can learn about the key given y_i we can also learn given y_{i-1} (e.g., we can erase one letter from y_{i-1} and guess the value of the key). Hence, for every probability distribution of Y_{i-1}, there is probability distribution of Y_i such that $I(Y_i, Z) \leq I(Y_{i-1}, Z)$, and therefore, the inequality $\inf_{Y_i} I(Y_i, Z) \leq \inf_{Y_{i-1}} I(Y_{i-1}, Z)$ will always hold.

The final lower bound (Theorem 2.1(4)) is a bound on the security that we can achieve for a given key length. For the probability of successful substitution, it holds that $\log P_S \geq -H(Z|Y_0)$ (see Theorem 5 [102]). Now, we have

$$\begin{aligned} P_d^2 &\geq P_I P_S \geq 2^{-\inf_{Y_t} I(Y_t,Z)} 2^{-H(Z|Y_0)} \\ &= 2^{I(Y_0,Z)-\inf_{Y_t} I(Y_t,Z)-H(Z)} \\ P_d &\geq \frac{2^{\frac{1}{2}(I(Y_0,Z)-\inf_{Y_t} I(Y_t,Z))}}{2^{\frac{1}{2}H(Z)}} \\ &\geq \frac{2^{\frac{1}{2}(I(Y_0,Z)-\inf_{Y_t} I(Y_t,Z))}}{\sqrt{|\mathcal{E}|}} \\ &\geq \frac{1}{\sqrt{|\mathcal{E}|}} \end{aligned} \tag{2.1}$$

■

2.3 Distance properties

The Hamming distance[4] between the messages is not important in the traditional authentication model because it is assumed that the messages arrive intact. However, the distances[5] between the messages of an η-code play a crucial role since they determine the erasure-tolerant aspects of the code. The following theorem describes several distance properties of the erasure-tolerant authentication codes.

Theorem 2.2 (Distance properties) *1. If the distance $d(x_1, x_2)$ between two plaintexts x_1 and x_2 is greater than r, then the distance $d(y_1, y_2)$ between the corresponding ciphertexts $y_1 = e_z(x_1)$ and $y_2 = e_z(x_2)$ is greater than t.*

2. If there is a code (set of codewords) $\zeta \subseteq \mathcal{S}$ whose distance $d(\zeta)$ is greater than r, then, for any encoding rule e_z, there is a code $\varsigma \subseteq \mathcal{M}_0(e_z)$ such that $|\varsigma| \geq |\zeta|$ and $d(\varsigma) > t$, where $\mathcal{M}_0(e_z)$ is the set of valid messages in $\mathcal{M}_0$ when the encoding rule e_z is in use.

[4]The number of letters that need to be changed in one message to obtain the other.
[5]Hereafter, when we say distance we mean Hamming distance.

3. *For each encoding rule e_z of a full-recovery η-code, there is a code $\varsigma \subseteq \mathcal{M}_0(e_z)$ such that $|\varsigma| \geq |\mathcal{S}|$ and $d(\varsigma) > t$.*

4. *For each encoding rule e_z of a deterministic full-recovery η-code, the distance of the code $\mathcal{M}_0(e_z)$ is greater than t.*

5. *Let $p_s = \min_{y_1,y_2 \in \mathcal{M}_0} P(y_2 \text{ valid } | y_1 \text{ valid})$. If p_s is greater than zero and the erasure-tolerant code is deterministic and full-recovery, then the distance between any two elements of $\mathcal{M}_0$ is greater than t.*

Proof.

1. Assume that $d(y_1, y_2) \leq t$. Let y be the message derived from y_1 (resp., y_2) by erasing the letters where y_1 and y_2 differ and let $x = e_z^{-1}(y)$ be the damaged plaintext corresponding to y. The plaintext x should be derivable from both x_1 and x_2 by erasing at most r letters. Therefore, the distance $d(x_1, x_2)$ is not greater than r, which is in contradiction with our assumption that $d(x_1, x_2) > r$.

2. Let us consider the code ς constructed as follows. For each codeword x in ζ, we put a codeword $y = e_z(x)$ in ς. Distance property of Theorem 2.2.1 implies that the distance of the code ς is greater than t. Clearly, the number of codewords in ς is equal to the number of codewords in ζ, and the theorem follows.

3. The third property follows from the previous property and the following observation. In the case of a full-recovery code, we have $r = 0$ and the set of all possible source states $\mathcal{S}$ forms a code with distance greater than r.

4. If the η-code is deterministic, then there is exactly one valid message given a source state and an encoding rule. If in addition the code is full-recovery, then the distance between any two intact valid messages should be greater than t.

5. The fact that p_s is greater than zero implies that for any two distinct messages y_1 and y_2 in $\mathcal{M}_0$, there is an encoding rule e_z such that y_1 and y_2 are both valid when the encoding rule e_z is used. According to the previous property, the distance between y_1 and y_2 must be greater than t.

■

2.4 Concatenation and composition of η-codes

In this subsection, we are going to consider two common construction techniques: concatenation and composition.

In the case of *concatenation*, the set of source states $\mathcal{S}$ of the new erasure-tolerant authentication code consists of all possible concatenations $x_1||x_2$, where $x_1 \in \mathcal{S}'_0$ is a source state of the first code and $x_2 \in \mathcal{S}''_0$ is a source state of the second code. To encode a source state $x = x_1||x_2$, we select two encoding rules $e_{z_1} \in \mathcal{E}'$ and $e_{z_2} \in \mathcal{E}''$, and compute the message y as $y = e_{z_1}(x_1)||e_{z_2}(x_2)$. It is not hard to verify that if we erase $t = t_1 + t_2 + 1$ letters from a message y, then we can "lose" at most $r = \max(r_1 + k_2, r_2 + k_1)$ letters of the corresponding source state x. Note that we allow more than t_1 (resp., t_2) letters to be erased from $e_{z_1}(x_1)$ (resp., $e_{z_2}(x_2)$). In that case, we check the validity of the second (resp., first) part of the message and discard the letters of the first (resp., second) part of the source state.

For the *composition construction*, the set of source states $\mathcal{S}''$ of the second code is equal to the set $\mathcal{M}'_0$ of all possible intact messages of the first code. The set of source states of the new code $\mathcal{S}$ is equal to the set of source states $\mathcal{S}'$ of the first code, and the set $\mathcal{M}_0$ of all possible intact messages is equal to the set $\mathcal{M}''_0$ of all intact messages of the second code. The message y corresponding to a given source state x is computed as $y = e_{z_2}(e_{z_1}(x))$, where the encoding rules $e_{z_1} \in \mathcal{E}'$ and $e_{z_2} \in \mathcal{E}''$ are chosen independently according to the corresponding probability distributions. We require r_2 to be less or equal than t_1, and if we erase at most $t = t_2$ letters from a message y, then we can "lose" at most $r = r_1$ letters of the corresponding source state x. Some relations between the impersonation and substitution probabilities of the new code and the impersonation and substitution probabilities of the component codes are provided below.

Theorem 2.3 *The following relations hold for the probability of successful deception of a concatenation and composition of η-codes:*

1. Concatenation: $P_d = \max(P'_d, P''_d)$

2. Composition: $P_I \le P'_I P''_I$, $P_S \le \tilde{P}'_S P''_S$, *where*

$$\tilde{P}'_S = \frac{\max_{y',y'' \in \mathcal{M}'} P((y', y'') \text{ valid})}{\min_{y \in \mathcal{M}'} P(y \text{ valid})}.$$

Proof.

1. Let us consider the message $y = y_1||y_2$, where $y_1 \in \mathcal{M}'$ and $y_2 \in \mathcal{M}''$. Since the encoding rules are chosen independently, we have

$$P(y \text{ valid}) = P(y_1 \text{ valid})P(y_2 \text{ valid}).$$

In that case, the probability $P(y \text{ valid})$ is less or equal than both P'_I and P''_I. Assume now that y_2 is not in $\mathcal{M}''$. The verifier will check only the validity of y_1. In this case, the probability that y is valid is $P(y \text{ valid}) = P(y_1 \text{ valid})$. Clearly, the probability $P(y \text{ valid})$ is again less or equal than P'_I, but we can select y_1 so that $P(y \text{ valid}) = P'_I$. Similarly, if we select y_1 so that there are more than t_1 erasures in it, the verifier will check the validity of y_2 only. The probability $P(y \text{ valid})$ will be less or equal than P''_I, and we can select y_2 so that $P(y \text{ valid}) = P''_I$. Therefore, the probability of successful impersonation of the new code is maximum of the probabilities of successful impersonation of the component codes (i.e., $P_I = \max(P'_I, P''_I)$).

An analogous argument holds for the probability of successful substitution. Let $y = y_1||y_2$ be the message intercepted by the adversary and let $\hat{y} = \hat{y}_1||\hat{y}_2$ be its substitution. Let us consider the case when $P'_S \geq P''_S$. It is not hard to show that the best substitution strategy is to substitute only the first part of the message (i.e., $\hat{y}_2 = y_2$ or $\hat{y}_2$ is derived by erasing more than t_2 letters). The probability that $\hat{y}$ is valid is equal to the probability that $\hat{y}_1$ is valid, and the probability of successful substitution P_S is equal to P'_S. Similarly, in the case when $P'_S \leq P''_S$, we get that $P_S = P''_S$. Therefore, $P_S = \max(P'_S, P''_S)$ and $P_d = \max(P'_d, P''_d)$.

2. First, we will show that $P_I \leq P'_I P''_I$. Let $y \in \mathcal{M}$ be an arbitrary message. For the probability that y is valid, we have

$$\begin{aligned}
P(y \text{ valid}) &= \sum_{z_1 z_2} \chi''(y, z_2)\chi'(e_{z_2}^{-1}(y), z_1)P(z_2)P(z_1) \\
&= \sum_{z_2} \chi''(y, z_2)P(z_2) \sum_{z_1} \chi'(e_{z_2}^{-1}(y), z_1)P(z_1) \\
&\leq P'_I \sum_{z_2} \chi''(y, z_2)P(z_2) \\
&\leq P'_I P''_I.
\end{aligned}$$

From the last inequality, it follows that $P_I = \max_{y \in \mathcal{M}} P(y \text{ valid}) \leq P'_I P''_I$.

Now, let us consider the conditional probability $P(y'' \text{ valid} \,|\, y' \text{ valid})$, where y' and y'' are two distinct messages from $\mathcal{M}$. We have

$$\begin{aligned}
P(y'' \text{ valid} \,|\, y' \text{ valid}) &= \frac{P((y', y'') \text{ valid})}{P(y' \text{ valid})} \\
&= \frac{\sum_{z_1 z_2} \phi''(y', y'', z_2)\phi'(e_{z_2}^{-1}(y'), e_{z_2}^{-1}(y''), z_1)P(z_2)P(z_1)}{\sum_{z_1 z_2} \chi''(y', z_2)\chi'(e_{z_2}^{-1}(y'), z_1)P(z_2)P(z_1)} \\
&\leq \frac{\max_{y'_1, y''_1 \in \mathcal{M}'} P((y'_1, y''_1) \text{ valid}) \times \sum_{z_2} \phi''(y', y'', z_2)P(z_2)}{\sum_{z_2} \chi''(y', z_2)P(z_2) \sum_{z_1} \chi'(e_{z_2}^{-1}(y'), z_1)P(z_1)} \\
&\leq \frac{\max_{y'_1, y''_1 \in \mathcal{M}'} P((y'_1, y''_1) \text{ valid}) \times \sum_{z_2} \phi''(y', y'', z_2)P(z_2)}{\min_{y'_1 \in \mathcal{M}'} P(y'_1 \text{ valid}) \times \sum_{z_2} \chi''(y', z_2)P(z_2)} \\
&\leq \tilde{P}'_S P''_S.
\end{aligned}$$

From the previous inequality, it is obvious that $P_S \leq \tilde{P}'_S P''_S$.

■

Note that authentication codes and erasure-resilient codes can also be used as component codes. Namely, authentication codes form a class of η-codes whose members provide authenticity, but no erasure-resilience ($t = 0$). The erasure-resilient codes on the other hand form a class of η-codes whose members provide erasure-resilience, but no authentication ($P_d = 1$).

Finally, we will consider the case when the probabilities $P((y', y'') \text{ valid})$ and $P(y \text{ valid})$ are uniformly distributed. In this case, the deception probability is characterized by the following corollary.

Corollary 2.4 *If the probabilities $P((y', y'') \text{ valid})$ and $P(y \text{ valid})$ $(y, y', y'' \in \mathcal{M}')$, are uniformly distributed, then the approximation $\tilde{P}'_S$ is equal to P'_S, and we have*

$$P_d \leq P'_d P''_d.$$

Proof. If the probabilities $P((y', y'') \text{ valid})$ and $P(y \text{ valid})$ are uniformly distributed, then

$$\tilde{P}'_S = \frac{\max_{y', y'' \in \mathcal{M}'} P((y', y'') \text{ valid})}{\min_{y \in \mathcal{M}'} P(y \text{ valid})} = \frac{P((y_1, y_2) \text{ valid})}{P(y_1 \text{ valid})} = P(y_2 \text{ valid} \,|\, y_1 \text{ valid})$$

where $y_1, y_2 \in \mathcal{M}'$ are two arbitrary messages. On the other hand, for the probability of successful substitution we have

$$P'_S = \max_{y_1, y_2 \in \mathcal{M}'} \frac{P((y_1, y_2) \text{ valid})}{P(y_1 \text{ valid})} = P(y_2 \text{ valid} \,|\, y_1 \text{ valid}).$$

Hence, $\tilde{P}'_S$ is equal to P'_S, and $P_S \leq P'_S P''_S$. From the last inequality and Theorem 2.3(2), it follows that $P_d \leq P'_d P''_d$. ■

2.5 η-codes with minimal impersonation and substitution probabilities

Not all η-codes can be represented as a composition of an authentication code and an erasure-resilient code. However, the members of one interesting class of erasure-tolerant authentication codes can always be represented as a composition of an authentication code and an erasure-resilient code since the messages in $\mathcal{M}_0$ form a code whose distance is greater than t. This is the class of η-codes whose probabilities of successful impersonation and substitution are minimal.

Theorem 2.5 *An η-code without secrecy (resp., with secrecy) has probability of successful impersonation $P_I = \frac{|\mathcal{S}|}{|\mathcal{M}_0|} < 1$ and probability of successful substitution $P_S = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $P_S = \frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$) if and only if*

1. *$d(\mathcal{M}_0) > t$ and*
2. *the underlying authentication code is an authentication code without secrecy (resp., with secrecy) such that $P_{uI} = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ and $P_{uS} = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $P_{uS} = \frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$).*

Proof. It is not difficult to show that if $d(\mathcal{M}_0) > t$, then $P_I = P_{uI}$ and $P_S = P_{uS}$. Clearly, if the underlying authentication code is without secrecy (resp., with secrecy), then the erasure-resilient code is without secrecy (resp., with secrecy) also.

Now, suppose that we have an erasure-resilient code without secrecy (resp., with secrecy) such that $P_I = \frac{|\mathcal{S}|}{|\mathcal{M}_0|} < 1$ and $P_S = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $P_S = \frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$). Since $P_I \geq P_{uI} \geq \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ and $P_S \geq P_{uS} \geq \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $P_S \geq P_{uS} \geq \frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$), we have $P_{uI} = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ and $P_{uS} = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $P_{uS} = \frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$). Therefore, the underlying authentication code has same impersonation and substitution probabilities as the erasure-tolerant authentication code. Obviously, if the erasure-tolerant authentication code is without secrecy (resp., with secrecy), then the underlying authentication code is without secrecy (resp., with secrecy) also.

It remains to be shown that $d(\mathcal{M}_0)$ is greater than t. For the underlying authentication code it holds that the probability $P(y \,\text{valid})$ is $\frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ for all $y \in \mathcal{M}_0$, and the probability $P(y_2 \,\text{valid} \,|\, y_1 \,\text{valid})$ is $\frac{|\mathcal{S}|}{|\mathcal{M}_0|}$ (resp., $\frac{|\mathcal{S}|-1}{|\mathcal{M}_0|-1}$) for any two distinct y_1 and y_2 in $\mathcal{M}_0$. Now, assume

that there are two messages y_1 and y_2 in $\mathcal{M}_0$ such that $d(y_1, y_2) \leq t$. Let y be the message in $\mathcal{M}$ derived by erasing all the letters in y_2 that differ from the corresponding letters in y_1. Since $P(y_2 \text{ valid} \,|\, y_1 \text{ valid})$ is less than 1, there is an encoding rule e_z such that y_1 is a valid message under the encoding rule e_Z, but y_2 is not a valid message under the encoding rule e_z. Therefore, we have $P(y \text{ valid}) > P(y_2 \text{ valid}) = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$, which is in contradiction with our assumption that $P_I = \frac{|\mathcal{S}|}{|\mathcal{M}_0|}$. ■

Stinson [106, 107, 108] has characterized the existence of authentication codes that have minimal impersonation and substitution probabilities in terms of existence of orthogonal arrays and balanced incomplete block designs (BIBD). Using these characterizations and Theorem 2.5, one can easily derive a relation between the existence of orthogonal arrays and BIBDs and the existence of η-codes with minimal P_I and P_S.

2.6 η-codes from set systems

A set system is a pair $(X, \mathcal{B})$ of a set $X = \{a_1, a_2, \ldots, a_k\}$ and a multiset $\mathcal{B}$ whose elements are subsets (or blocks) of X.

One can construct an η-code from a set system as follows. The set X will consist of the letters of the source state x. Then, we use an authentication code to compute an authentication tag for each block $B_i \in \mathcal{B}$. The message y is constructed by appending the authentication tags of the blocks to the source state x. Now, if some letter of the message is erased, and the erased letter does not belong to a block B_i or to the authentication tag of the block B_i, then we can still check the authenticity of the letters of the plaintext that belong to B_i.

2.6.1 Constructions from covering designs

One possible construction is from a complementary design of a covering. The set system $(X, \mathcal{B})$, where $|X| = k$, is a (k, m, t)-covering design if all blocks are m-subsets of X, and any t-subset of X is contained in at least one block. Some efficient constructions of coverings can be found in [82, 96]. The complementary set system of a set system $(X, \mathcal{B})$ is the set system $(X, \mathcal{B}^c)$, where $\mathcal{B}^c = \{X \backslash B_i | B_i \in \mathcal{B}\}$. It is not hard to prove the following property of the complementary design of a covering[6].

Lemma 2.6 *Let $(X, \mathcal{B})$ be a (k, m, t)-covering design. Then, for the complementary design, we have*

[6] A similar result was used in [34].

1. $|B_i| = k - m$ *for all* $B_i \in \mathcal{B}^c$

2. *For any subset* $F \subset X$ *such that* $|F| \leq t$, *there is a block* $B_i \in \mathcal{B}^c$ *such that* $F \bigcap B_i = \emptyset$.

The following proposition trivially follows from the previous lemma.

Proposition 2.7 *If at most* t *letters are erased from a message of an* η*-code derived from a complementary design of a* (k, m, t)*-covering, then we can verify the authenticity of at least* $k - m$ *letters of the plaintext* $(r \leq m)$.

Now, we are going to consider a specific example. Assume that the source state is a sequence of v^2 packets arranged in a square matrix

$$P_{0,0}, \ldots, P_{0,v-1}, \ldots, P_{v-1,0}, \ldots, P_{v-1,v-1},$$

where each packet $P_{i,j}$ is a sequence of l letters. We divide the set of blocks $\mathcal{B}$ into two disjoint subsets $\mathcal{R}$ and $\mathcal{D}$. The blocks in $\mathcal{R}$ are constructed from the rows of the matrix

$$\mathcal{R} = \{R_i | R_i = \{P_{i,0}, \ldots, P_{i,v-1}\}, 0 \leq i < v\}.$$

The blocks in $\mathcal{D}$ are "parallel" to the main diagonal

$$\mathcal{D} = \{D_i | D_i = \{P_{0,i}, P_{1,(i+1) \bmod v}, \ldots, P_{v-1,(i+v-1) \bmod v}, 0 \leq i < v\}.$$

The set system consisting of the set of all packets $P_{i,j}$ and the set of all blocks R_i and D_i is a complementary design of a $(v^2, v(v-1), v-1)$-covering. That is, if at most $v - 1$ packets are lost, then, we can still verify the authenticity of at least one block (i.e., v packets). The set system also has the following convenient properties:

- If one packet is lost, then we can still verify the authenticity of all $v^2 - 1$ packets that are not lost.

- If two packets are lost, then we can still verify the authenticity of at least $v^2 - 4$ packets.

Table 2.2 compares the example presented here to the case when each packet is authenticated separately. The complexities are expressed in number of multiplications over the finite field when the Horner's procedure is used for polynomial evaluation. We can see that in the example presented here, the number of keys (the part that changes) is significantly smaller when the number of packets increases. The price we pay is slightly increased time complexity and smaller erasure-tolerance.

Table 2.2: Using $(v^2, v(v-1), v-1)$-covering vs Authentication of each packet separately

code	multiplicative complexity	number of keys	erasures tolerated
from $(v^2, v(v-1), v-1)$-covering	$v^2 l + 2v + \log l - 1$	$2v$	up to $v-1$
each packet separately	$v^2 l$	v^2	up to $v^2 - 1$

2.6.2 Constructions from cover-free families

A cover-free family is defined as follows.

Definition 2.1 [110] *Let X be a v-set and let $\mathcal{B}$ be a set of subsets (blocks) of X. The set system $(X, \mathcal{B})$ is called a (w,t)-cover-free family if for any w blocks $B_1, \ldots, B_w \in \mathcal{B}$ and any other t blocks $A_1, \ldots, A_t \in \mathcal{B}$, we have*

$$\bigcap_{i=1}^{w} B_i \nsubseteq \bigcup_{j=1}^{t} A_j. \tag{2.2}$$

Let $N(w,t,b)$ denote the minimum number of points in any (w,t)-cover-free family that has b blocks. The following bound can be found in [110]:

$$N(w,t,b) \leq \frac{(w+t)\log b}{-\log p}, p = 1 - \frac{t^t w^w}{(t+w)^{t+w}}. \tag{2.3}$$

Let us consider the following generic construction of erasure-tolerant unconditionally secure authentication codes or erasure-tolerant signature schemes. Given a large message M, the sender divides it into a sequence of b smaller messages $M_1, \ldots, M_b$ and computes v authentication tags (resp., signatures) $\tau_1, \ldots, \tau_v$. Each tag τ_j is computed over a subsequence of the sequence of messages using a message authentication (or signature) scheme. We say that the authentication tag τ_j depends on the message M_i if M_i is in the subsequence which is used to compute τ_j. The sender then constructs and sends b packets $P_1, \ldots, P_b$. Each packet P_i includes the message M_i and all authentication tags τ_j with the following property: the message M_i is the last message in the subsequence that is used to compute τ_j.

We allow at most t of the b packets to be lost during the transmission. The recipient uses the received tags to verify the authenticity of the received messages. In particular, if all messages that are used to compute some authentication tag τ_j have arrived, the recipient uses the received τ_j to verify the authenticity of the subsequence. If the verifier outputs one, then the recipient accepts all the messages in the subsequence as valid.

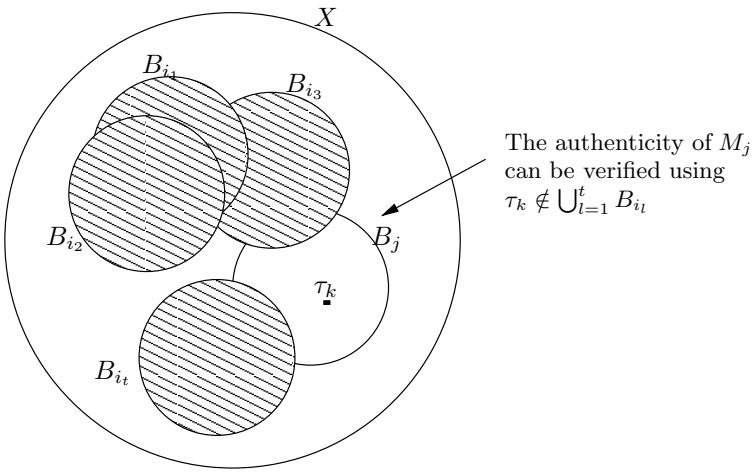

Figure 2.1: Erasure-tolerant authentication using cover-free families

Obviously, if a tag τ_j depends on a lost message M_i, then τ_j cannot be used to verify the authenticity of the rest of the messages in the subsequence that was used to compute τ_j. Hence, a situation might occur where the recipient will not be able to verify the authenticity of a message that was not lost. If the subsequences are chosen in such manner so that the receiver will be able to verify the authenticity of each message that is not lost, then we say the erasure-tolerant authentication code (resp., signature scheme) is *non-degrading*.

To analyze the problem of constructing non-degrading schemes, it is convenient to define a *dependency set system* $(X, \mathcal{B})$, where:

- $X = \{\tau_1, \ldots, \tau_v\}$
- $\mathcal{B} = \{B_j | B_j = \{\tau_i | \tau_i \text{ depends on } M_j\}, 1 \leq j \leq v\}$

If a message M_i is lost during the transmission, then the authentication tags in B_i become useless. Assume that $M_{i_1}, \ldots, M_{i_t}$ are the lost messages. Then, the set of all useless tags is $\bigcup_{l=1}^{t} B_{i_l}$ (see Figure 2.1). The authenticity of the message M_j can be verified if and only if $B_j \nsubseteq \bigcup_{l=1}^{t} B_{i_l}$. Hence, we have the following theorem.

Theorem 2.8 *An erasure-tolerant authentication code (resp., signature scheme) constructed as above is non-degrading if and only if the corresponding dependency set system is a* $(1, t)$*-cover-free family.*

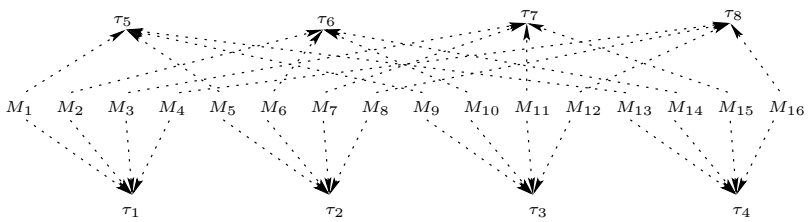

Figure 2.2: $(1, 1)$-cover-free family

A simple example is given in Figure 2.2. One can easily verify that if one message is lost ($t = 1$), then we can still verify the authenticity of the rest of the packets. For example, assume M_1 is lost. Then, the authentication tags τ_1 and τ_5 are not usable anymore. However, the authenticity of M_2, M_3 and M_4 still can be verified using τ_6, τ_7 and τ_8 respectively, and the authenticity of M_5, M_9 and M_{13} can be verified using τ_2, τ_3 and τ_4 respectively.

Note that the number of tags in the example is eight instead of sixteen, which is the case when we authenticate each packet separately. In general, we have two extremes. The number of tags can be as low as a logarithm of the number of packets (see inequality 2.3). The other extreme is to use one tag per message (i.e., the number of tags is equal to the number of messages). The main advantages of using a small number of tags / signatures are:

- The communication (or storage) overhead is smaller.
- When unconditionally secure message authentication is employed, we need a fresh key material for each tag. Hence, the key generation time decreases when the number of tags is smaller.
- In the case of digital signature schemes, the time complexity will be reduced since the number of signatures is smaller.

Remarks. To simplify the description, we assumed that the messages arrive in order. However, it might be possible that the messages can be reordered so that the derived sequence is also valid. A malicious reordering can be prevented if the messages also include a unique number.

Also, note that more than t erasures can be tolerated in a scheme derived from $(1, t)$-cover-free family. In particular, if more than t messages are lost, we can still verify the authenticity of some of the received messages. However, we cannot verify the authenticity of all received messages (i.e., non-degrading property does not hold). For example, in the scheme from Figure 2.2, even if three

messages are lost, we will still be able to verify the validity of at least four messages.

2.6.3 Efficiency issues

In the generic construction of an η-code from a set system, multiple authentication tags can depend on a single message. Authenticating each subsequence of messages anew can lead to large time complexity since each message is processed many times. Here, we present a more efficient solution.

We will use the following unconditionally secure multiple message authentication code [1]. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_v$ be a sequence of v messages. The authentication tag for a message $\mathbf{a}_i$ is computed as

$$h_{x,y,z_i}(\mathbf{a}_i) = y(a_{i_0} + a_{i_1}x + \ldots + a_{i_{l-1}}x^k) + z_i = yf_{\mathbf{a}_i}(x) + z_i,$$

where $x, y, z_i, a_{i_0}, \ldots, a_{i_{l-1}} \in \mathbf{F}_q$ (q a prime power). The key parts x and y remain unchanged for all messages in the sequence. Only the part z_i is refreshed for each message.

We process each message only once and produce a sequence of b values $\alpha_1, \ldots, \alpha_b$ (see Figure 2.3. The temporary values α_i are then combined to produce the authentication tags.

The temporary values α_i are computed as

$$\alpha_i = f_{M_i}(x) = m_0 + m_1x + \ldots + m_{l-1}x^{l-1},$$

where $m_0, \ldots, m_{l-1} \in \mathbf{F}_q$ are the letters of the message M_i.

Given a subsequence $M_{i_0}, \ldots, M_{i_n}$ of messages, the authentication tag that depends on these messages can be efficiently computed as:

$$\tau = y(f_{M_{i_0}}(x) + x^l f_{M_{i_1}}(x) + \ldots + x^{nl} f_{M_{i_n}}(x)) + z_i.$$

In other words, we evaluate a polynomial over each message separately, and then combine the results to evaluate polynomials over subsequences of messages. An efficient procedure for polynomial evaluation is given in [1]. The time complexity of the procedure is 7-13 machine instructions per word.

2.7 η-codes from Reed-Solomon codes

Reed-Solomon codes have already been used to construct authentication codes [10]. Here, we present a construction of η-codes based on Reed-Solomon codes.

Let $a_1, a_2, \ldots, a_k$ be the sequence of plaintext letters, where each a_i is an element of the Galois field F_q. The message is constructed by appending $n - k$ authentication tags $\tau_1, \tau_2, \ldots, \tau_{n-k}$

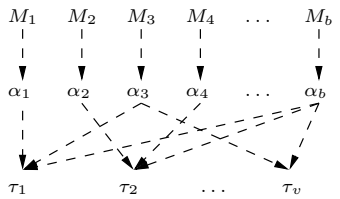

Figure 2.3: Efficient computation

($\tau_i \in F_q, 1 \leq i \leq n-k$) to the plaintext. The authentication tags are computed as

$$\tau_i = y_i + \sum_{j=1}^{k} a_j x_i^j$$

where x_i, y_i are randomly selected and secret (i.e., part of the key). In addition, we require that $x_{i_1} \neq x_{i_2}$ for $i_1 \neq i_2$.

Now, assume that t_a letters of the plaintext and t_τ authentication tags are lost during the transmission. Note that $t_a + t_\tau$ cannot be greater than t. The receiver can use t_a of the received authentication tags and the $k - t_a$ received letters of the plaintext to construct a system of linear equations that can be always solved for the unknown letters of the plaintext. Once the missing letters of the plaintext are recovered, the receiver can check the authenticity using the remaining $n - k - t_a - t_\tau$ authentication tags. Assuming that each letter of the message can be erased with same probability p, the probability that in a sequence of v messages, there is no forged message accepted as valid, is lower bounded by the product

$$\prod_{i=0}^{t} (1 - \frac{k^{n-k-i}}{q^{n-k-i}})^{v \times \binom{n}{i} p^i (1-p)^{n-i}}.$$

The authentication tags in the example have dual role. They can be used to verify the authenticity of the plaintext or to recover some lost letters of the plaintext. Therefore, the η-codes described above, offer more security than the codes constructed by composing an authentication code and an erasure-resilient code on one hand, and they are more resilient to erasures than authentication codes on the other hand. This is illustrated in Table 2.3. The first row corresponds to a code that is derived by composing an authentication code and an erasure-resilient code that can recover one erasure. The second row corresponds to a code of length $n = k+2$ constructed as above. Since only one erasure is tolerated, the condition $x_{i_1} \neq x_{i_2}$ for $i_1 \neq i_2$ is not necessary, and the complexity can be reduced from $2k$ to k multiplications by using a multiple message authentication

Table 2.3: Comparison of an η-code from RS code with the current practices

code	multiplicative complexity	non-deception probability	erasure tolerance
composition	k multipl. in F_q	$(1-\frac{k}{q})^v$	1 erasure
our η-code	k multipl. in F_q	$(1-\frac{k}{q})^{pv}(1-\frac{k^2}{q^2})^{(1-p)v}$	1 erasure
A-code	$k/2$ multipl. in F_{q^2}	$(1-\frac{k}{2q^2})^v$	0 erasures

code as in the previous example. The final code is an ordinary authentication code. The letters of the message in the last case are considered to be elements of a Galois field F_{q^2}.

CHAPTER 3

Proven Secure Stream Authentication in a Point-to-point Setting

In this chapter, we adapt the notions of existential forgery and unforgeability [53] to the case of peer-to-peer stream authentication, and introduce the notions of stream authentication tree and stream authentication forest in order to describe the tag computation process of a general class of peer-to-peer stream authentication schemes. The analysis of the security of tree and forest stream authentication schemes shows that one can color (assign tag computation procedures to) different nodes of the tree so that the resulting scheme is secure regardless of how the trees are constructed. We use the results of the analysis to provide proofs of security for three new practical stream authentication schemes: MACC, ReMAC and SN-MAC.

3.1 Unforgeable Stream Authentication

In a stream signature scheme, the data sequence is divided into smaller pieces we refer to as chunks, and an authenticator is assigned to each chunk. The decision whether a particular chunk is accepted as authentic is made based on the portion of the stream up to that chunk and the authenticator associated with that chunk. Thus, the delay between the moment when the chunk is received and the moment when it is accepted is reduced allowing for the receiver to consume the incoming bits almost as they arrive.

We must be able to distinguish between the end of the stream and the state that data is continuing to arrive. This strategy allows the receiver to know whether the adversary has shortened the message by deleting the last chunks. Note that *this security property is not achieved by Gennaro-Rohatgi* [43].

A stream signature scheme consists of a key generation algorithm G, a signing algorithm S and a verifying algorithm V. The key generation algorithm G outputs a pair of keys (s, p). The signing algorithm S takes as input a key s and a stream $\mathbf{M} = (M_1, \ldots, M_l)$. Based on the key s and

$(M_1, \ldots, M_i)$ it computes a μ_i. We call the l-tuple $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_l)$ the *stream signature* of $\mathbf{M}$. $\mu_i (i \leq l)$ is called a *temporary signature*.

On input a key p, $(M_1, \ldots, M_i)$, and $(\mu_1, \ldots, \mu_i)$, algorithm V outputs i type/authenticity pairs $((b_1, v_1), \ldots, (b_i, v_i))$, where the *type bits* $b_j \in \{0, 1\}$ are used to indicate whether the corresponding chunk is the last chunk in the stream or not, and the *authenticity bits* $v_j \in \{0, 1\}$ indicate whether the chunk is valid (accepted) or not. We say that the string $M_1 || \ldots || M_i$ is *accepted as partial stream* if the stream $M_1 || \ldots || M_{i-1}$ is accepted as a partial stream, the type bit b_i is zero and the authenticity bit v_i is one (authentic). The string $M_1 || \ldots || M_i$ is *accepted as complete stream* if $M_1 || \ldots || M_{i-1}$ is accepted as a partial stream (or i equals one), the type bit b_i is one and the authenticity bit v_i is one. If the stream $M_1 || \ldots || M_i$ is not accepted, then it is *rejected*. It is obvious that if the stream $M_1 || \ldots || M_i$ is rejected or accepted as complete, then the streams $M_1 || \ldots || M_j (i < j \leq l)$ are rejected. A formal definition follows.

Definition 3.1 (Stream Signature Scheme) *A* stream signature scheme *is a triple* (G, S, V) *of probabilistic polynomial-time algorithms satisfying the following conditions:*

1. *The algorithm* G *(called the* key generator*) outputs a pair of bit strings* (s, p).

2. *On input a string* s *and* $(M_1, \ldots, M_i)$*, where* $M_i \in \{0, 1\}^+$ *the* signing algorithm S *outputs* $\mu_i \in \{0, 1\}^+$*. If the complete stream is* $\mathbf{M} = (M_1, \ldots, M_l)$ *we call* $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_l)$ *the* stream signature*, where* $l \in \{1, 2, 3, \ldots\}$.

3. *On input a string* p*,* $(M_1, \ldots, M_i)$ *and* $(\mu_1, \ldots, \mu_i)$ $(M_i, \mu_i \in \{0, 1\}^+)$*, the* verification algorithm V *outputs* i type/authenticity pairs (b_1, v_1), $\ldots$, (b_i, v_i), $b_j, v_j \in \{0, 1\}$*. If there is a* $k \in \{1, 2, \ldots, i\}$ *such that* $b_k = 1$ *or* $v_k = 0$*, then the authenticity bits* $v_j (k < j \leq l)$ *are zero, where* l *is the number of chunks* M_i *in the complete stream* $\mathbf{M} = (M_1, \ldots, M_l)$.

4. *For every pair* (s, p) *in the range of* G*, and for every* $\mathbf{M} = (M_1, \ldots, M_l), M_i \in \{0, 1\}^+, l \in \{1, 2, 3, \ldots\}$*, algorithms* S *and* V *satisfy*

$$\forall i : 1 \leq i < l \quad \Pr[V(p, \mathrm{Pref}_i(\mathbf{M}), \mathrm{Pref}_i(S(s, \mathbf{M}))) = (\underbrace{(0,1), (0,1), \ldots, (0,1), (0,1)}_{i-\text{pairs}})] = 1$$

$$\Pr[V(p, \mathbf{M}, S(s, \mathbf{M})) = (\underbrace{(0,1), (0,1), \ldots, (0,1), (1,1)}_{l-\text{pairs}})] = 1$$

where $\mathrm{Pref}_i((x_1, x_2, \ldots, x_l)) = (x_1, x_2, \ldots, x_i)$*. The probability is taken over the internal coin tosses of* S *and* V.

The previous definition says nothing about the security of the scheme. In order to prove that a particular stream signature scheme is secure, we need first to define what it means for a stream signature scheme to be secure. We will follow the existential forgery and exact security approaches [5, 53] and we will consider the *private key (stream authentication)* case only. The security of a public key stream signature scheme can be defined in a similar manner. The only difference is that the adversary in a public key scheme has access to the verification key p.

An adversary for a stream authentication scheme is a probabilistic algorithm E with oracle access to a signing and a verifying algorithm for a random but secret key pair (s, p). The adversary can request a stream signature of an arbitrary stream $\mathbf{M}$ by writing $\mathbf{M} = (M_1, \ldots, M_l)$ on a special query tape. The signing oracle computes the signature $\boldsymbol{\mu}$ and returns it to E. E can also ask the verifier whether $\boldsymbol{\mu}$ is a valid signature for $\mathbf{M}$ by writing $(\mathbf{M}, \boldsymbol{\mu})$ on a special query tape. The verifying oracle returns an l-tuple $((b_1, v_1), \ldots, (b_l, v_l))$. We will assume that there is some limit L on the size of the streams that can be submitted to the signing and verification oracles.

The goal of the adversary is to trick the receiver into accepting some string $M_1||\ldots||M_k$ as a partial or complete stream when either the string was never signed before or it was signed before as a different type (e.g, it was signed as a complete stream, but now it is accepted as partial). In the first case, the adversary can change the contents of the stream. In the second case, by changing the type, the adversary can either cut the stream or trick the receiver into believing that the stream is longer than its actual size. We keep track of the strings that are already signed using a *set of queries* $Q_E^{S(s,\cdot)}$. For each signing query $(M_1, M_2, \ldots, M_l)$, we add the stream/type pairs $(M_1, 0)$, $\ldots$, $(M_1||\ldots||M_{l-1}, 0)$, $(M_1||\ldots||M_l, 1)$ to the set of queries $Q_E^{S(s,\cdot)}$. The adversary E is successful if it makes a verify query $((M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l))$ such that $M_1||M_2||\ldots||M_k$ is accepted (as partial or complete) for some $k \in \{1, 2, \ldots, l\}$, and $(M_1||\ldots||M_k, b_k)$ not in the set of queries. In other words, the output of the verifier is $((0,1),(0,1),\ldots(b_k,1),(b_{k+1},v_{k+1}),\ldots,(b_l,v_l))$, but $(M_1||\ldots||M_k, b_k) \notin Q_E^{S(s,\cdot)}$. The pair $((M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l))$ is called a *forgery* and the k-tuple $((M_1, \mu_1), \ldots, (M_k, \mu_k))$ is called a *partial forgery*[1].

The attack on the stream authentication scheme is (q_s, q_v)-attack if the adversary makes no more than q_s signing queries and no more than q_v verify queries. If, in addition, E runs for no more than t steps, then the (q_s, q_v)-attack is a (t, q_s, q_v)-attack. We say that the adversary $[q_s, q_v, \epsilon]$-breaks the scheme if the attack is a (q_s, q_v)-attack and it is successful with at least ϵ probability. The adversary $[t, q_s, q_v, \epsilon]$-breaks the scheme if the attack is (t, q_s, q_v)-attack and the probability

[1] The term partial forgery is somewhat misleading since we use it even when the stream $M_1||\ldots||M_k$ is complete.

of success is at least ϵ. The scheme is $[t, q_s, q_v, \epsilon]$-unforgeable if there is no adversary that can $[t, q_s, q_v, \epsilon]$-break it. The formal definition is given below.

Definition 3.2 (Unforgeability) *The stream authentication scheme (G, S, V) is* $[t, q_s, q_v, \epsilon]$- unforgeable *if for every probabilistic algorithm E that runs in at most t steps, and makes at most q_s queries to a signing oracle $S(s, \cdot)$ and at most q_v queries to a verification oracle $V(p, \cdot)$, it holds that*

$$\begin{aligned} \Pr[\exists i \in \{1, \ldots, l\} \text{ s.t.} \quad & V(p, (M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l)) = ((0,1), \ldots, (b_i, 1), (b_{i+1}, v_{i+1}), \ldots, (b_l, v_l)) \\ \wedge \quad & (M_1 || \ldots || M_i, b_i) \notin Q_E^{S(s,\cdot)} \ ] \ < \ \epsilon \end{aligned}$$

where $b_j, v_j \in \{0,1\}, 1 \leq j \leq l$, (s, p) is a key pair generated by G, $((M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l))$ is E's output and $Q_E^{S(s,\cdot)}$ is the set of queries. The probability is taken over the coin tosses of G, S and V as well as over the coin tosses of E.

3.2 Some Practical Schemes

In this section we present three practical schemes. We prove their security in Section 3.4.5.

3.2.1 SN-MAC

SN-MAC (sequence numbers and MACs) is a straightforward stream authentication scheme. There is a unique number N_s assigned to each stream (e.g., using counter). A sequence number i, which determines the position of the chunk within the stream, and a type bit b_i, which indicates whether the chunk is the last chunk of the stream or not, are assigned to each chunk of the stream. The chunks are then signed (authenticated) independently using some message authentication scheme (see Fig 3.1).

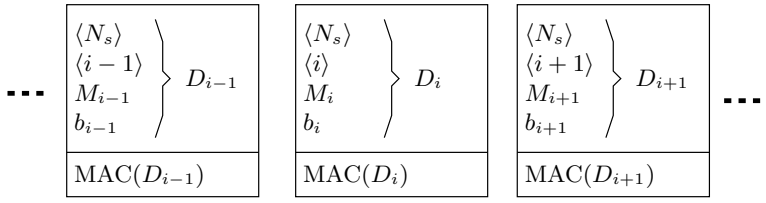

Figure 3.1: SN-MAC

Suppose that the data D_i in each packet i consists of three blocks $D_i = d_{i,1}||d_{i,2}||d_{i,3}$, and that a CBC-like MAC scheme (e.g., OMAC) is used for the computation of the authentication tags. In this case, the generation of the authentication tag MAC(D_i) can be described using a labeled tree as in Fig 3.2. The blocks of D_i are assigned as labels to the external nodes (leaves) of the tree. A message label msg[x] and a tag tag[x] are assigned to each internal node x. The message label msg[x] is a concatenation of the messages corresponding to the children of x, and it "keeps track" about what part of the stream is processed to get the tag tag[x]. The tag tag[x] is either a function of the message label msg[x] (e.g.,tag[1], tag[2] and tag[4]) or a function of the tags of the children of x (e.g., tag[3] and tag[5]). Note that only tag[5] will be used for the verification of the authenticity of D_i. The rest of the tags are some intermediate results.

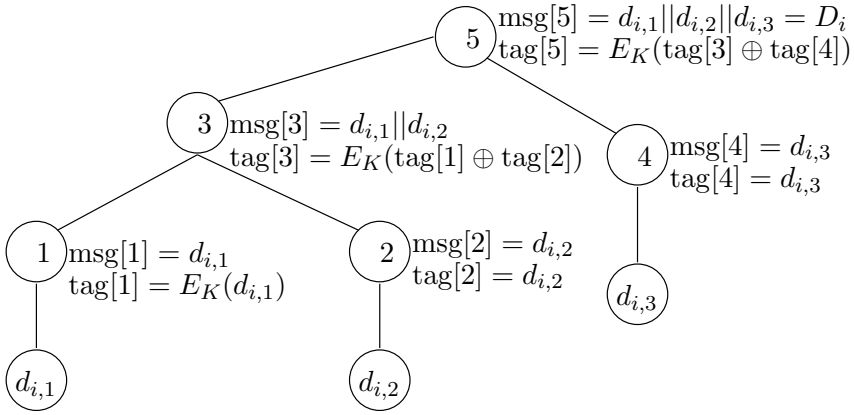

Figure 3.2: A tree-based description of the computation of MAC(D_i) in SN-MAC

3.2.2 ReMAC

In a ReMAC (recompute the MAC) scheme, the temporary signature associated with some chunk M_i is computed by signing the partial stream $M_1||\ldots||M_i$, not just the chunk M_i (see Fig 3.3).

The tree-based description of ReMAC is given in Fig 3.4.

3.2.3 MACC

MACC (MAC chaining) scheme is depicted in Fig 3.5. An authentication tag is computed for each chunk M_i of the stream M and then the chunk M_i is chained to the previous chunk by computing MAC of the concatenation of the tags. It is assumed that σ_0 is an empty string.

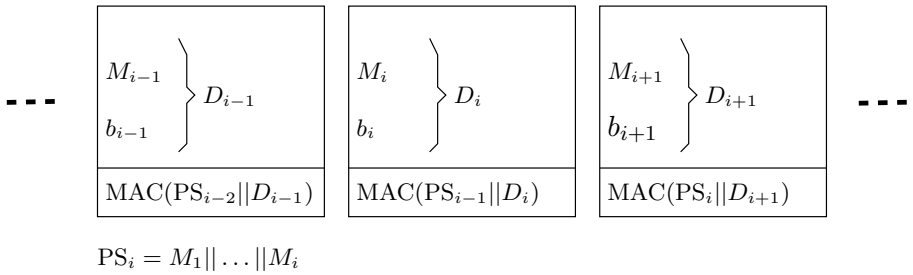

Figure 3.3: ReMAC; $\text{PS}_i = M_1||\ldots||M_i$.

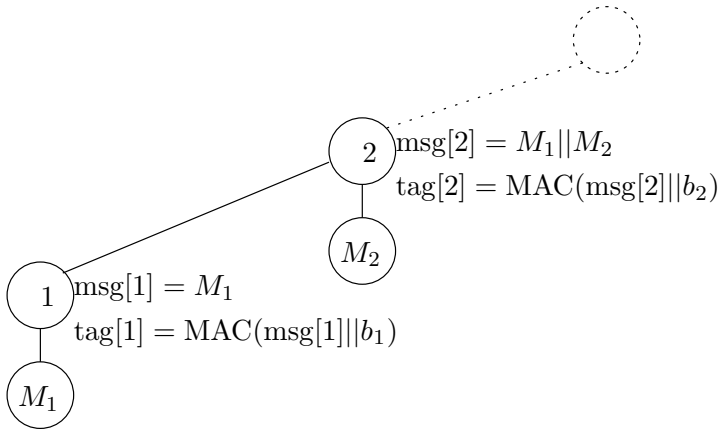

Figure 3.4: A tree-based description of the computation of authentication tags in ReMAC

A tree-based description of the tag generation in MACC is given in Fig 3.6.

3.2.4 Comparison of ReMAC, MACC and SN-MAC

Each of the previously described schemes has advantages and disadvantages depending on the application and the properties of the underlying MAC scheme. The main advantage of the SN-MAC scheme is that the temporary signatures depend only on the particular chunk, and we can check the authenticity of the chunk even if some of the packets are lost. This is convenient in applications where packet loss is tolerable (e.g., audio, video, etc.). The disadvantages of the scheme are that we need to provide mechanisms that will guarantee (with high probability) the uniqueness of the stream number N_s, and there is a computational overhead due to the processing of the stream and sequence numbers when computing the authentication tags.

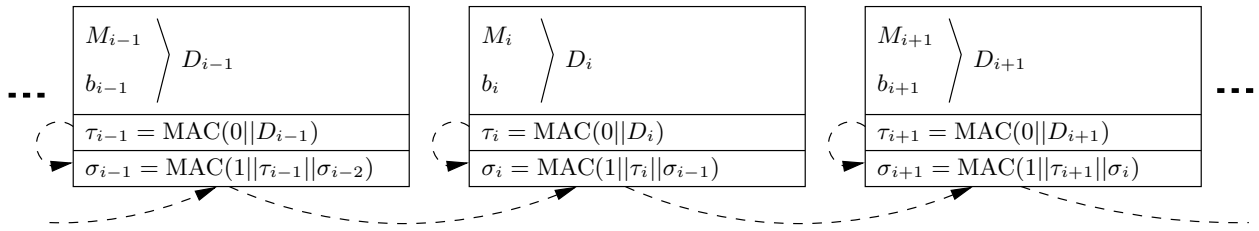

Figure 3.5: MACC

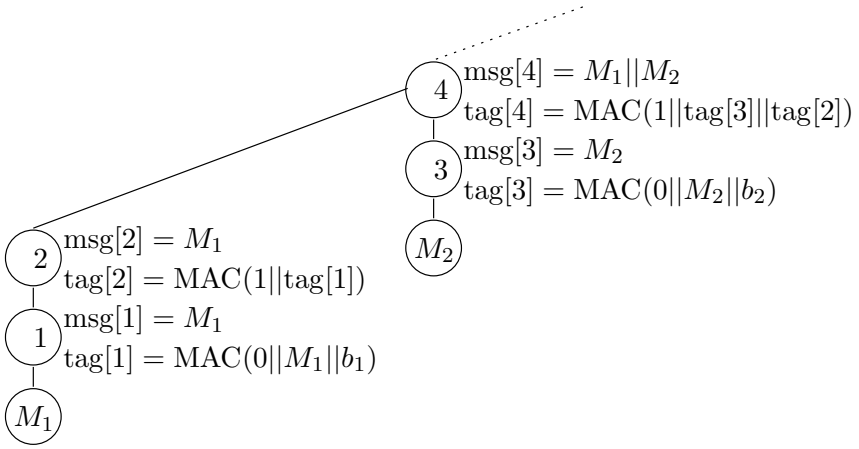

Figure 3.6: A tree-based description of the computation of authentication tags in MACC

At first glance, it seems that the time complexity of the ReMAC scheme is quadratic in the length of the stream. However, by storing some values when computing the temporary signature μ_{i-1}, we can efficiently compute the temporary signature μ_i, that is, the performance of ReMAC is not worse than the performance of SN-MAC and MACC even when the underlying MAC scheme is not incremental. When an incremental MAC scheme (e.g., XMACR [6]) is used, the performance of ReMAC is slightly better than the performance of SN-MAC and MACC. Clearly, if a chunk is lost in a ReMAC scheme, we cannot verify the authenticity of the subsequent chunks, and the scheme can be used only for applications that are not erasure-tolerant.

The MACC scheme is another stateless scheme, and it can be advantageous over the ReMAC scheme in some scenarios. For example, consider a sequence of messages $M_1, M_2, \ldots, M_l$ that are exchanged between two or more parties. We can view these messages as chunks of a stream

and use a stream authentication scheme to ensure their authenticity. Furthermore, assume that the underlying MAC scheme is not incremental (e.g., OMAC [57]). A ReMAC scheme cannot be efficiently implemented in this scenario. As previously mentioned, the efficient computation of a temporary signature μ_i in a ReMAC scheme relies on the knowledge of some value that has been derived during the computation of μ_{i-1}. However, in our scenario, the parties that compute μ_{i-1} and μ_i can be different.

The rate at which we can authenticate the stream data is an upper bound on the achievable stream bit rate. We can increase this rate by computing the temporary signatures in a distributed manner. One possible generalization of the MACC scheme is to divide the chunk into several smaller pieces. An authentication tag is computed for each piece by a different node in the distributed network, and the computed tags are than combined to derive the final tag that is sent along with the chunk. Assume now that the underlying MAC scheme is not parallelizable, and that we want to implement the SN-MAC scheme in a distributed manner. In this case, we need to reduce the size of the chunks leading to increased communication overhead. A MACC-like scheme would be slightly more efficient in this setting. Using a ReMAC scheme in this setting is also less efficient since we need to protect the messages exchanged between the nodes against disclosure.

The disadvantage of MACC is that it is not erasure-resilient.

3.3 A family of schemes

In this section, we generalize the design principles of the schemes introduced in Section 3.2 by introducing the notions of stream authentication tree and stream authentication forest. This generalized approach is motivated by the following arguments:

- We can avoid the problem of providing a separate security proof for each scheme presented in Section 3.2 by proving the security in the general case. Furthermore, we can select other practical schemes from the general class discussed here for our specific application. The security of these schemes trivially follows from the security of the general class.

- There are other applications besides the typical stream applications like audio or video, where a sequence of chunks (or messages) needs to be authenticated (e.g., exchange of related messages). We can use the stream authentication schemes as another level of abstraction that can be used to design and prove security of more complex constructions using reduction techniques instead of other methods, like formal methods for security protocol verification

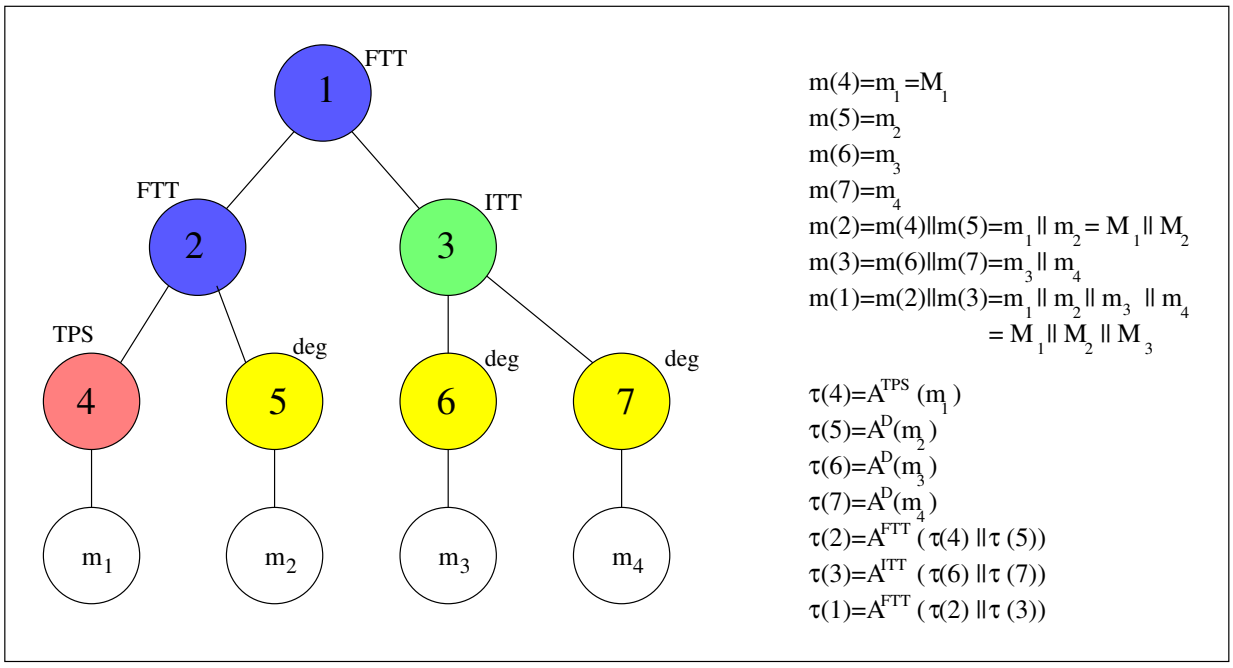

Figure 3.7: An example of a 4C stream authentication tree

[22, 100].

- Message authentication is just a special case of stream authentication where the stream consists of only one chunk. There are number of MAC schemes (e.g., [6, 16, 17, 57]) that can be described using the notion of stream authentication tree introduced in this paper.

- The MAC schemes can be used to build secure stream authentication schemes. Therefore the analysis of the stream authentication schemes can help us better understand the desirable properties of MAC schemes. For example, the analysis in Section 3.4.3 shows that it is important if a MAC schemes is forgeable on collision or not. Whether the ReMAC scheme can be efficiently implemented or not depends on the underlying MAC scheme, etc.

An example of a stream authentication tree is shown in Fig 3.7. The pieces of the stream are assigned as message labels to the external nodes (leaves) of the tree. A message label $m(x)$, a tag $\tau(x)$, a tag type and an algorithm label A_x are assigned to each internal node x. We now explain this in more details.

The message $m(x)$ is a concatenation of the messages corresponding to the children of x starting from the first (left) child and ending with the last child. So, $m(x)$ is the concatenation of the pieces of the stream assigned to the leaves of the subtree rooted at x when traversed in postorder (postfix). We say that the external node (leaf) x in a tree T is a *leftmost leaf* iff every ancestor of x (including x itself) is the first (left) node among its siblings. The *leftmost path* is the path between the root and the leftmost leaf. The internal nodes in the leftmost path have a special role. Namely, the messages associated with some of these nodes are the partial (or complete) streams to be authenticated, and the corresponding tags are final results of some temporary signature computation.

We split the tag types based on two properties, being:

Whether the tag type is final tag or not. A *final tag* is defined as one that can be used to verify the authenticity of a (partial) stream. Otherwise it is called *intermediate*.

Whether the tag is a tag-of-tags, or a tag-of-chunks. We allow two ways to compute the tag $\tau(x)$ associated with an internal node x, either

- as a function of the message corresponding to x or,
- as a function of the tags of the children of x.

So, in total we can distinguish between 4 tag types, or internal nodes in the tree. We can now categorize the nodes of the tree as following:

Definition 3.3 Message nodes *are the external nodes of the tree. A piece of the stream is assigned as a message label to each message node and the concatenation of the message labels of the external nodes when visited in postorder gives the stream.*

Internal nodes *can be categorized as:*

Degenerated *Such a node is intermediate and is a tag-of-chunks.*

TPS (tag-of-partial-stream) *Such a node is a final node and is a tag-of-chunks.*

ITT (intermediate-tag-of-tags) *Such a node is intermediate and is a tag-of-tags.*

FTT (final-tag-of-tags) *Such a node is final tag and is a tag-of-tags.*

Definition 3.4 (Stream Authentication Tree) Stream authentication trees *are rooted ordered trees. A message label is assigned to each external node (leaf) of the tree. A 4-tuple*

$(m(x), \tau(x), t(x), A_x)$ is assigned to each internal node x, where the string $m(x)$ is a message label, the string $\tau(x)$ is a tag, $t(x) \in \{\text{degenerated}, \text{TPS}, \text{ITT}, \text{FTT}\}$ is a tag type, as defined in Definition 3.3, and A_x is an algorithm label that specifies the tag computation algorithm used at the node x. Let $v_1, v_2, \ldots, v_k$ be the children of an internal node x. The following properties must hold for the message and tag labels:

$$m(x) = m(v_1)||m(v_2)||\ldots||m(v_k)$$

$$\tau(x) = \begin{cases} A_x(m(x)) & \text{if } t(x) \in \{\text{degenerated}, \text{TPS}\} \\ A_x(\tau(v_1)||\tau(v_2)||\ldots||\tau(v_k)) & \text{if } t(x) \in \{\text{ITT}, \text{FTT}\} \end{cases}$$

Note that the input to the tag computation algorithm is the concatenation of tags $\tau(v_1)||\ldots||\tau(v_k)$ instead of a tuple $(\tau(v_1), \tau(v_2), \ldots, \tau(v_k))$. Such definition is more convenient for the analysis presented below since all tag computation algorithms have exactly one input. We don't think that our choice is too restrictive since the individual tags can be extracted from the concatenated string (e.g., using known or fixed tag length).

Lemma 3.1 *The internal nodes have the following properties:*

1. *If x is a parent of an external node (leaf), then the tag type $t(x)$ cannot be* ITT *or* FTT.
2. *If x is not in the leftmost path, then the tag type $t(x)$ cannot be* TPS *or* FTT.

Proof. The first property follows straightforward from the definition since ITT and FTT are both tag-of-tags. The second follows from the fact that a partial stream must have the chunk M_1 in it. ■

In a stream authentication scheme based on the notion of a stream authentication tree, the signing algorithm first constructs a tree and assigns types and algorithms to the nodes of the tree, and then, it computes the authentication tags. A more formal description is given below. In most of the practical schemes, the structure of the authentication tree and the algorithms that will be used for the computation of the tags are known in advance, and they are not changed over time. Therefore, one can compute the authentication tags without having to construct a stream authentication tree.

We will refer to the labeled tree derived from a stream authentication tree by discarding the message, tag and algorithm labels of the internal nodes as the *structure* of the stream authentication

tree. For each chunk M_i of the stream $(M_1, \ldots, M_l)$, the signing algorithm of a *tree stream authentication scheme* first runs a (probabilistic) structure construction procedure. The structure construction procedure takes as input the structure constructed to compute the temporary signature for the partial stream $M_1||\ldots||M_{i-1}$, the length of the chunk and information indicating whether the chunk is last or not, and outputs the "new" part of the structure that will be used to compute the temporary signature of the partial stream $M_1||\ldots||M_i$. Clearly, the tag type of the new root must be either TPS or FTT, and the tag type of any other new node must be either degenerated or ITT. After the structure construction phase, the signing algorithm runs a (probabilistic) coloring procedure. The coloring procedure assigns algorithm labels to the new nodes. As mentioned before, the algorithm labels specify which algorithm will be used to compute the tag. The algorithm labels are selected from some set of algorithm labels (or colors) that is previously agreed upon between the sender and the receiver. The labels in the set can refer to both algorithms that use key and algorithms that do not use a key. If an algorithm label refers to an algorithm that uses a key, then it must specify the actual key that is used. For example, $\mathrm{CBC}-\mathrm{MAC}_{K_0}$ is a possible algorithm label and it specifies that the tag will be computed using CBC-MAC when the key is K_0. After the structure construction and the coloring, the signing algorithm computes the message and tag labels of the new nodes. Obviously, the newly constructed tree should be such that the message label of its root is $M_1||\ldots||M_i$. The temporary signature of the partial stream $M_1||\ldots||M_i$ consists of the tags of the new nodes[2] and the randomness used by the structure construction procedure, the coloring procedure and the tag computation algorithms.

Given the stream and the temporary signatures, the verification algorithm reconstructs the tree. The message label of a TPS or FTT node is accepted as valid only if the computed tags equal the tags that were sent and the previous partial streams are also valid. The set of all strings is partitioned into a set of complete and a set of partial streams (e.g., using end-of-text character). If the message label belongs to some predefined set of complete streams, then it is accepted as a complete stream. Otherwise, it is accepted as a partial stream.

The concept of stream authentication tree can be extended to stream authentication forest.

Definition 3.5 (Stream Authentication Forest) *A* stream authentication forest *is an ordered*

[2]In order to speed-up the stream authentication, one can compute the tags in a distributed manner where each node computes a tag and sends the result to some other node. Since we have defined the temporary signature to include all "new"tags (not just the final tag), a distributed implementation of a secure scheme will remain secure even if the adversary can eavesdrop the communication between the nodes. However, in practice, we can send only the final tags and the randomness needed by the verifier to recompute them.

forest of stream authentication trees.

In a *forest scheme*, a unique number N_s is assigned to each stream that is submitted for signing. A prefix to the message label of the leftmost leaf of each tree is also added. The prefix is a concatenation of the binary representations of the stream number N_s and the position j of the tree within the forest (analogously to SN-MAC scheme). We will refer to the scheme used to construct the trees of the forest as an underlying tree scheme.

3.4 Security Analysis

In this section, we examine the security properties of the tree and forest stream authentication schemes. We show that we need at least four colors (different algorithms) in order to achieve security regardless of the structure of the stream authentication trees.

3.4.1 A particular approach

In Section 3.3 we did not specify restrictions on A_x. To facilitate proving general results we will now consider a simpler case. We will restrict A_x such that $A_x = A^{t(x)}$. Since we have four tag types, we could consider that we color the algorithms A_x based on these tag types.

We are primarily interested in deriving algorithms for $A^{t(x)}$ such that we can prove the resulting scheme to be secure regardless of the structure construction procedure that was used in the scheme.

In order to provide an answer, we consider the class of 4C (four color) schemes. An example of a 4C stream authentication tree is given in Fig 3.7. 4C schemes color the different node types (degenerated, TPS, ITT and FTT) using different colors. We denote by A^D, A^{TPS}, A^{ITT} and A^{FTT} the tag computation algorithms used at the degenerated, TPS, ITT and FTT nodes respectively. We will show that 4C forest schemes have *forgery or collision property* (defined further on), that is, if there is an adversary that can construct a forgery for the 4C forest scheme, then there is an adversary that can forge a tag or find a collision for some of the tag computation procedures A^{D}, A^{TPS}, A^{ITT} or A^{FTT}.

As usual, we assume that all algorithms are known to the adversary. Only the secret keys used by the sender and the receiver are unknown. From the discussion above, it follows that given the stream, the tags and the randomness, the adversary can reconstruct the stream authentication tree except for the secret keys that are used by the tag computation algorithms. Hence, hereafter, we slightly abuse the security model of the previous section and assume that the verify queries and the responses to signing queries are *shadowed* stream authentication trees that are constructed as

follows. Each algorithm label that refers to an algorithm that uses a key is replaced by a label that specifies only the family, not the key. For example, the label $\mathrm{CBC}-\mathrm{MAC}_{K_0}$ in the stream authentication tree will become CBC − MAC in the shadowed tree. We say that the shadowed tree rooted at a node x is *valid* if $m(x)$ is accepted when the tree is submitted to the verifier. The valid shadowed tree T is a *partial forgery tree* if the partial (resp., complete) stream corresponding to the root of the tree was not signed as partial (resp., complete) stream during some previous signing query. Let's consider a valid shadowed tree T. There is some input associated with each tag in the tree. For example, if the tag is a tag of a degenerated node, then the corresponding input is the message label of the node. If the tag is a tag of an ITT node, then the corresponding input consists of the tags of the children of the node, etc. The set of all such input/tag pairs is called a *set of authentic pairs corresponding to* T. The union of all sets of authentic pairs corresponding to trees constructed during some previous signing queries is called a *pool of known authentic pairs*. If an authentic input/tag pair is not in the pool of known authentic pairs, then we say that it is a *forgery pair*. Two input/tag pairs *collide* if the tags are equal but the inputs or the algorithms used to compute the tags are different. Given a node x in some stream authentication forest, a *stream authentication subforest at* x is the forest consisting of all the trees preceding the tree containing x and the subtree rooted at x. Let $((M_1, \mu_1), \ldots, (M_i, \mu_i))$ be some partial forgery and let x be the node where $M_1 || \ldots || M_i$ is accepted. The subforest at x is called a *partial forgery forest* with *final node* x. Now, we can formally express the forgery or collision property.

Lemma 3.2 (Forgery or Collision Property) *Let P_f be the set of authentic pairs corresponding to some 4C partial forgery forest consisting of the trees $T_1, \ldots, T_k$. At least one of the following statements is true:*

1. *There is a tree $T_i \in \{T_1, \ldots, T_k\}$ such that the tag associated with its root is forged.*
2. *There is a pair in P_f that collides with some known authentic pair.*

Proof. We will first establish some relations between the ability of constructing partial forgery trees in 4C tree schemes and the ability of constructing forgeries and collisions for the underlying tag computation procedures. The attribute 4C will be omitted in this section. For example, when we say partial forgery tree, we mean partial forgery tree in a 4C scheme. Let's start with the case when the partial forgery tree is rooted at a TPS node. The result is trivial but we need to state it formally for later reference.

Lemma 3.3 *If T is a partial forgery tree rooted at a TPS node x, then the pair $(m(x), \tau(x))$ is a forgery pair.*

Proof. Assume that the pair $(m(x), \tau(x))$ is not a forgery pair. Then, the tag $\tau(x)$ was already computed by applying the procedure A^{TPS} on input $m(x)$ during some previous signing query. From the definition of partial forgery tree and the fact that $m(x)$ was already signed during some previous query it follows that T is not a partial forgery tree. ■

When the partial forgery tree is rooted at an FTT node x, the tag associated with x does not have to be forged. It is possible to construct partial forgery by finding collisions. The following lemma provides result about the collision properties of the sets of authentic pairs corresponding to two distinct valid trees whose roots have equal tags.

Lemma 3.4 *Let T_1 and T_2 be two distinct valid trees with FTT roots u_0 and v_0 correspondingly such that $\tau(u_0) = \tau(v_0)$ and $m(u_0) \neq m(v_0)$. Then, there are a pair $\mathbf{p_1}$ in the set of authentic pairs corresponding to T_1 and a pair $\mathbf{p_2}$ in the set of authentic pairs corresponding to T_2 such that $\mathbf{p_1}$ and $\mathbf{p_2}$ collide.*

Proof. Let us consider the recursive procedure **collision(T_1,T_2)**:

1. If the roots u_0 and v_0 have different number of children, then return 1.
2. If the tag of some child of u_0 is different from the tag of the corresponding child of v_0, then return 1.
3. If the color of some child of u_0 is different from the color of the corresponding child of v_0, then return 1.
4. If there is a child x of u_0 and a child y of v_0 such that x and y are both TPS or both degenerated and $m(x) \neq m(y)$, return 1.
5. If there are two distinct subtrees T_1' (rooted at u_1, a child of u_0) and T_2' (rooted at v_1, the corresponding child of v_0) such that $\tau(u_1) = \tau(v_1)$, $m(u_1) \neq m(v_1)$, and u_1 and v_1 are either both ITT or both FTT, then return **collision(T_1',T_2')**. Otherwise, return 0.

First, we will prove that the procedure always returns 1 when the trees T_1 and T_2 satisfy the conditions of the lemma. This is done in two steps: by proving that the procedure can not return 0, and by proving that the procedure can not run forever. Next, we show that a return value of 1 implies an existence of collision pairs.

Let $u_0, u_1, \ldots$ and $v_0, v_1, \ldots$ be the sequences of roots of the trees in the recursive procedure calls. The procedure will return zero only if there is some t so that none of the collision conditions (Steps 1-4) is satisfied and there is no node u_t (a child of u_{t-1}) and a corresponding node v_t (a child of v_{t-1}) with the properties required in the last step of the procedure.

We will now show that, when T_1 and T_2 satisfy the assumptions of the lemma and none of the collision conditions is satisfied, the subtrees T_1' and T_2' in step 5 can always be constructed. If none of the collision conditions is satisfied, then u_0 and v_0 must have same number of nodes, and the tag and the color of any child of u_0 are equal to the tag and the color of the corresponding child of v_0. Since the message labels of u_0 and v_0 are different, there is a node u_1, a child of u_0, such that $m(u_1) \neq m(v_1)$, where v_1 is the child of v_0 that corresponds to u_1. Note that the colors and the tags of u_1 and v_1 must be same. The nodes u_1 and v_1 can not be degenerated or TPS because we assumed that the collision condition 4 is not satisfied. Hence, the nodes u_1 and v_1 are either both colored ITT or both colored FTT.

The analysis above can be extend to arbitrary u_t and v_t in a straightforward manner. Let u_{t-1} and v_{t-1} satisfy the requirements of step 5 ($\tau(u_1) = \tau(v_1)$, $m(u_1) \neq m(v_1)$, u_1 and v_1 are either both ITT or both FTT). Again, if the collision conditions are not satisfied for the trees rooted at u_{t-1} and v_{t-1}, then u_{t-1} and v_{t-1} must have same number of nodes, and the tag and the color of any child of u_{t-1} are equal to the tag and the color of the corresponding child of v_{t-1}. Since the message labels of u_{t-1} and v_{t-1} are different, there is a node u_t (a child of u_{t-1}) and a node v_t (the corresponding child of v_{t-1}) such that $m(u_t) \neq m(v_t)$. Note that the colors and the tags of u_t and v_t must be same. The nodes u_t and v_t can not be degenerated or TPS because we assumed that the collision condition 4 is not satisfied. Hence, the nodes u_t and v_t are either both colored ITT or both colored FTT. Hence, if the collision conditions are not satisfied, we can always find two subtrees that satisfy the requirements for recursive call in the fifth step. Furthermore, the depth of the node u_t is t and it can not be larger than the height of the tree. Hence, the procedure will finish in finite number of steps and it will return a value. According to the previous discussion, that value cannot be 0.

It is not difficult to see that if one of the collision condition is satisfied, then we can construct

collision pairs. Since the procedure returns 1 if and only if one of the collision conditions is satisfied, it follows that for any valid trees that satisfy the assumptions of the lemma one can find collision pairs in the sets of authentic pairs corresponding to these trees. ■

The following lemma gives a relation between the set of authentic pairs corresponding to some 4C partial forgery tree and the set of known authentic pairs. More precisely, the set of authentic pairs corresponding to a partial forgery tree contains a forgery pair or a pair that collides with some of the known authentic pairs.

Lemma 3.5 *Let T be a partial forgery tree with a root x. At least one of the following statements is true:*

1. *$\tau(x)$ is forged.*
2. *There is a pair in the sets of authentic pairs corresponding to T that collides with some known authentic pair.*

Proof. If the root x of the partial forgery tree T is colored TPS, then by Lemma 3.3, the tag $\tau(x)$ is forged.

Let us consider the case when the root is FTT. Let $v_1, \ldots, v_k$ be the children of x. Assume that T is a partial forgery tree and the pair $((\tau(v_1)|| \ldots ||\tau(v_k)), \tau(x))$ is not a forgery pair, but a known authentic pair. Then, there is a valid tree T' generated as a result of some previous signing query such that $\tau(x) = \tau(x')$, where the FTT node x' is the root of T'. The message labels of the nodes x and x' must be distinct. Otherwise, the tree T is not a partial forgery tree. According to Lemma 3.4, there is a pair in the set of authentic pairs corresponding to T that collides with some pair in the set of authentic pairs corresponding to T' which is a subset of the set of all known authentic pairs. Hence, at least one of the statements must be true. ■

The following relation between partial forgery forests and the partial forgery trees will help us to extend the previous result to the case of 4C forest schemes.

Lemma 3.6 *Let x be a final node of some partial forgery forest consisting of the trees $T_1, \ldots, T_k$. At least one of trees $T_1, \ldots, T_k$ is a partial forgery tree for the underlying 4C tree scheme.*

Proof. First, we will consider the case when there is no known stream signature with the same stream number as the partial forgery forest. Then, the message label of the final node x is unique. Namely, the message $m(x)$ begins with the stream number and no message that begins with that number has been signed by the underlying 4C tree scheme. Therefore, the tree T_k is a partial forgery tree for the underlying tree scheme.

Now, assume that there is some previous signing query (there can be only one such query) whose stream number is same as the stream number of the partial forgery forest $T_1, \ldots, T_k$. Let $T'_1, \ldots, T'_n$ be the trees of the forest corresponding to that signing query. If the number of trees n is smaller than k, then again the message label $m(x)$ is unique (has unique tree number) and the tree T_k is a partial forgery tree for the underlying tree scheme. Finally, assume that $n \geq k$ and the tree rooted at x is not a partial forgery tree for the underlying tree scheme. This means that there is some node y in T'_k with the same message label as x. In that case, the forest $T_1, \ldots, T_k$ can be a partial forgery forest only if $k > 1$ and $M_1 || \ldots || M_{k-1} \neq M'_1 || \ldots || M'_{k-1}$, where M_i denotes the part of the stream associated with the tree T_i (derived from the message label of the root of T_i by discarding the prefix consisting of the stream number and the tree number) and M'_i denotes the part of the stream associated with the tree T'_i. Hence, there is an index $i < k$ such that M_i and M'_i are different. Since M_i and M'_i are different, the message labels $m(x_i)$ and $m(x'_i)$, where x_i is the root of T_i and x'_i is the root of T'_i, must be different also. Therefore, the stream $m(x_i)$ has never been signed by the underlying tree scheme and the tree T_i is a partial forgery tree for the underlying tree scheme. ■

The forgery or collision property (Lemma 3.2) follows from Lemma 3.5 and Lemma 3.6. ■

3.4.2 Three colors for the internal nodes are not sufficient

From the discussion in Section 3.4.1, it is obvious that if we select the tag computation procedures so that it is hard to forge tags or find collision pairs, then the 4C forest scheme will be secure regardless of how we construct the trees. Moreover, the coloring scheme presented in Section 3.4.1 not only provides security but also uses minimal number of colors. We will demonstrate this by presenting an example of structure construction procedure with the following property: any scheme that uses the procedure and only three colors for the intermediate nodes can be broken.

Fig 3.8 depicts the possible structures of the stream authentication trees when the stream

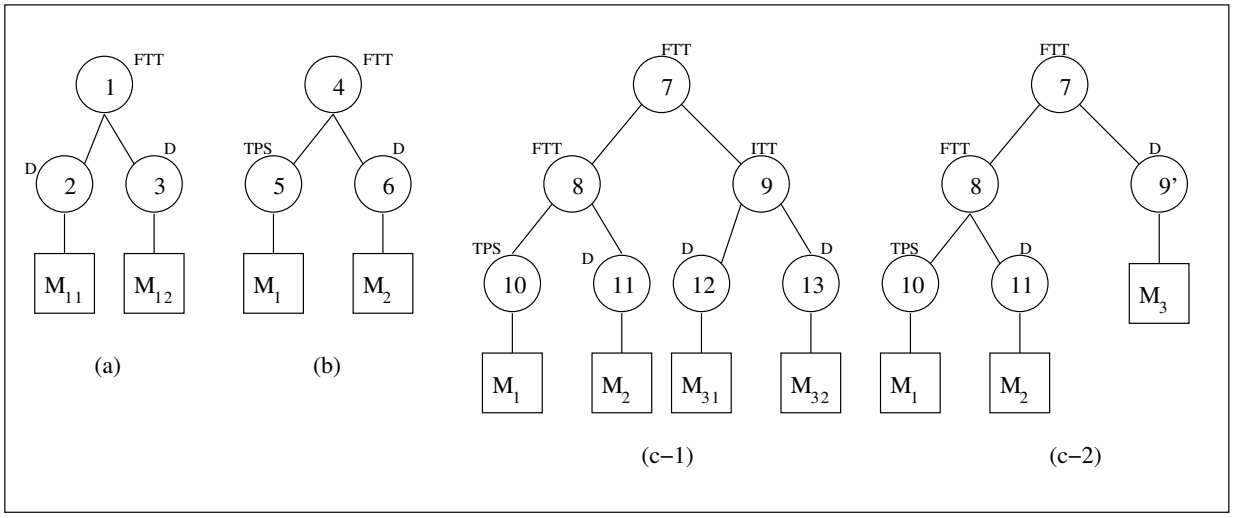

Figure 3.8: Possible structures when the stream consists of one, two or three chunks

consists of one (Fig 3.8.a), two (Fig 3.8.b) or three (Fig 3.8.c) chunks. When the stream consists only of one chunk M_1, the chunk is divided into two parts M_{11} and M_{12} that are assigned as labels to the message nodes. A tag for each part of the chunk is computed at the degenerated nodes 2 and 3. The tags $\tau(2)$ and $\tau(3)$ are then used to compute the final tag $\tau(1)$ at the FTT node. If the stream consists of two chunks, then the first chunk is used to compute the tag $\tau(5)$ at the TPS node. The tag $\tau(5)$ is a temporary signature for the partial stream M_1. The second chunk M_2 is used to compute the tag $\tau(6)$. The final tag $\tau(4)$ is computed from the tags $\tau(5)$ and $\tau(6)$. When the stream consists of three chunks, the structure construction procedure flips a coin. Depending on the outcome, the procedure either divides the last chunks into two chunks M_{31} and M_{32}, and constructs the structure depicted in Fig 3.8.c.1, or constructs the structure depicted in Fig 3.8.c.2.

We will show that any coloring of these structures that uses at most three colors for the internal nodes is not secure. Assume that there is a secure stream authentication scheme that uses the previously described structure construction procedure and only three colors. If that is the case, then the most probable colors of the nodes 4, 5 and 6 must be different. Assume that the most probable colors of the nodes 5 and 6 are equal. Then, the adversary can submit a stream $(M_1, M_2), (M_1 \neq M_2)$ for signing and construct a partial forgery tree as follows. Take the subtree rooted at the node 6 and change the type of the node 6 from degenerated to TPS. The probability of the attack is $\geq \frac{1}{3}$ (since the most probable color appears with probability $\geq \frac{1}{3}$). This is in

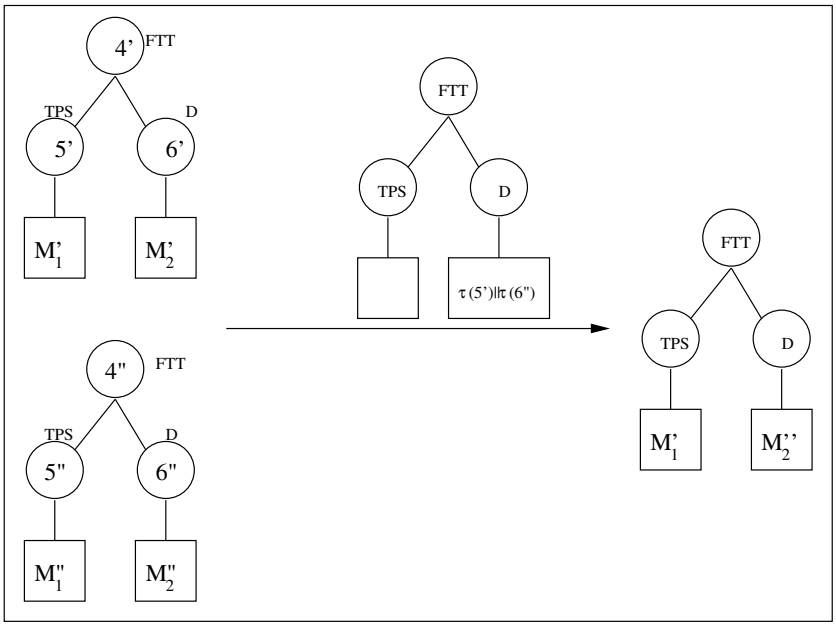

Figure 3.9: Constructing a forgery when 4 and 6 have same color

contradiction with our assumption that the scheme is secure. Assume now that the most probable colors of the nodes 4 and 5 are equal. The adversary can construct a partial forgery tree by discarding the message nodes, merging the nodes 5 and 6 into one message node whose label is $\tau(5)||\tau(6)$, and changing the type of 4 from FTT to TPS. The probability of success is large since the probability that $\tau(5)||\tau(6)$ will be equal to M_1 or $M_1||M_2$ for randomly selected M_1 and M_2 is very small. Otherwise, we will be able to break the scheme by guessing a node 5 tag. Again, this is in contradiction with our assumption that the scheme is secure, and thus, the most probable color of node 4 is different than the most probable color of 5. Finally, assume that the most probable color of 4 is equal to the most probable color of 6. Then we can use the degenerated node 6 to forge a node 4 tag. One possible construction of a partial forgery tree is depicted in Fig 3.9. Clearly, the success probability of the attack is high ($\geq \frac{1}{3}$), and, under our assumption that the scheme is secure, the most probable colors of 4 and 6 must be different.

An analogous analysis shows that all degenerated nodes must have same most probable color, and their most probable color must be different from the most probable colors of the TPS and FTT nodes. Similarly, all TPS nodes have same most probable color and it is different than the most probable color of the FTT and degenerated nodes. Now, we will show that if the most probable

color of the ITT node 9 is equal to the FTT (resp., degenerated or TPS) most probable color, then we can break the scheme. Hence, there is no secure 3-coloring for the structure construction procedure described above. If the most probable color of 9 is equal to the FTT most probable color, then we can construct a partial forgery tree by taking the subtree rooted at 9 and changing the type of 9 from ITT to FTT. The derived stream authentication tree will will be of the type depicted in Fig 3.8.a. If the most probable color of the node 9 is equal to the TPS most probable color, then the adversary can construct a partial forgery in the following manner. Take the subtree rooted at 9. Discard the message nodes. Merge the degenerated nodes 12 and 13 into a message node whose label is $\tau(12)||\tau(13)$. Change the tag type of 9 from ITT to TPS. Finally, if the most probable color of the node 9 equals the degenerated most probable color, then the adversary can construct a partial forgery tree by converting the structure c-1 into the structure c-2 as follows. Discard the message nodes whose labels are M_{31} and M_{32}. Merge the degenerated nodes 12 and 13 into one message node whose label is $\tau(12)||\tau(13)$. Change the type of 9 from ITT to degenerated. If M_3 is different than $\tau(12)||\tau(13)$, then the constructed tree is partial forgery tree. The probability of the attack is small only if the chunk M_3 is equal to $\tau(12)||\tau(13)$ with large probability. However, in this case, we can break the scheme using different attack. It is not hard to see that if M_3 equals $\tau(12)||\tau(13)$ with high probability, then most of the time, the tags computed at the degenerated nodes are equal to the input used to compute them. In this case, we can mount an attack similar to the one depicted in Fig 3.9. Namely, we can assign tags computed at nodes 5 and 6 as message labels to the children of 2 and 3, and forge a node 4 FTT tag.

3.4.3 Security based on unforgeable MACs

We will introduce some new property for MAC schemes (see Definition 3.6) that will allow to prove our results. Note that we demonstrate that certain MAC schemes satisfy this new property.

We are going to show that a secure (unforgeable) 4C forest scheme can be obtained if a secure MAC scheme is used for tag computation. In particular, the tag computation procedures that we are going to consider are implemented as:

$$\begin{aligned} A^{\mathrm{D}}(m(x)) &= \mathrm{MAC}(00||m(x)) \\ A^{\mathrm{TPS}}(m(x)) &= \mathrm{MAC}(01||m(x)) \\ A^{\mathrm{ITT}}(\tau(v_1)||\ldots||\tau(v_k)) &= \mathrm{MAC}(10||\tau(v_1)||\ldots||\tau(v_k)) \\ A^{\mathrm{FTT}}(\tau(v_1)||\ldots||\tau(v_k)) &= \mathrm{MAC}(11||\tau(v_1)||\ldots||\tau(v_k)) \end{aligned}$$

Definition 3.6 *A MAC scheme is* $[t, q_s, q_v]$-forgeable on collision *if there is an algorithm A that takes as input an initial finite set P of known authentic message/authenticator pairs and two pairs from P that collide, and outputs a forgery for the MAC scheme. The algorithm A runs in at most t time and it makes at most q_s signing and at most q_v verifying queries. If the two message/authenticator pairs are the only two pairs from P that collide, then the scheme is* $[t, q_s, q_v]$-forgeable on first collision.

Lemma 3.2 and the forgeability on collision property imply the following theorem.

Theorem 3.7 *Let E be an adversary that $[t', q'_s, q'_v, \epsilon]$-breaks a 4C forest scheme that uses a $[t'', q''_s, q''_v]$-forgeable on (first) collision MAC scheme for tag computation, and let $q = L(q'_s + q''_v + 1) + q''_s + q''_v$, where L is the maximum number of internal nodes that can appear in a forest. Then, there is an adversary that $[t' + t'' + cq, q, \epsilon]$-breaks (i.e., outputs a forgery) the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant.*

Proof. Suppose there is an adversary E that $[t', q'_s, q'_v, \epsilon]$-breaks a 4C forest scheme, and suppose that the underlying MAC is $[t'', q''_s, q''_v]$-forgeable on collision. We will construct an algorithm U that $[t' + t'' + cq, q, \epsilon]$-breaks the MAC scheme.

U's procedure is to run E and answer its oracle queries. In order to construct the answer to E's signing or verification query, U makes at most L queries to its oracle $\mathcal{O}$. U stores the result of each query in the memory[3] and checks for collision in a following manner. Let m be the message corresponding to the query, let τ be the tag corresponding to the query and let (m, τ) be valid. If there is no message stored at address $\tau_0 + \tau$, where τ_0 is some constant integer, U stores m at that location. If there is a message stored at address $\tau_0 + \tau$ and the message is equal to m, U doesn't store anything and continues. If there is a message stored at address $\tau_0 + \tau$ and the message is not equal to m, U invokes the algorithm A that constructs a forgery on collision. The A's input is the set of all pairs (m, τ) that are known authentic so far and the two pairs that collide. If no collision occurs when constructing the answers to E's queries, then E will finish and output its result. For each message/authenticator pair corresponding to the E's answer, U checks whether it is a forgery pair or it collides with some known authentic pair stored in the memory. If it is a forgery pair, U outputs it and halts. If it collides with some known authentic pair, then U invokes the algorithm A that produces a forgery on collision. Otherwise, U just halts.

[3] We use RAM computational model.

The number of queries that U submits to its oracle $\mathcal{O}$ is not greater than $q = L(q'_s + q'_v + 1) + q''_s + q''_v$. The time complexity of U is at most $t' + t'' + cq$, where t' is the time required to break the stream authentication scheme, t'' is the time required to find a forgery pair for the MAC when MAC collision occurred, and cq is the time required to construct the answers to the queries. The stream authentication scheme is breakable with at least ϵ probability and the probability of forging a MAC when the stream scheme is broken is 1. Hence, the probability of breaking the MAC is at least ϵ. ■

The following proposition gives an answer to the question whether secure MAC schemes that are forgeable on first collision exist.

Proposition 3.8 *PMAC and OMAC are $[c(|P| + l_M), 1, 0]$-forgeable on first collision, where l_M is the maximum message length, $|P|$ is the size of the set of known authentic pairs and $c > 0$ is a small implementation dependent constant.*

Proof. Let $m_1 = m_1[1]||\dots||m_1[n_1 - 1]||m_1[n_1]$ and $m_2 = m_2[1]||\dots||m_2[n_2 - 1]||m_2[n_2]$ be two messages whose authenticators are identical when computed by the signing algorithm of the PMAC [17] (resp., OMAC [57]) scheme. The blocks $m_1[1], \dots, m_1[n_1 - 1]$ of the message m_1 and the blocks $m_2[1], \dots, m_2[n_2 - 1]$ of the message m_2 are of length l_B (the block length). The last blocks $m_1[n_1]$ and $m_2[n_2]$ can have length less than l_B and the length of $m_1[n_1]$ can be different from the length of $m_2[n_2]$.

By considering various cases for the possible lengths of $m_1[n_1]$ and $m_2[n_2]$, one can show that we can always compute $m'_1 = m_1[1]||\dots||m_1[n_1 - 1]||m'_1[n_1]$ and $m'_2 = m_2[1]||\dots||m_2[n_2 - 1]||m'_2[n_2]$ so that:

1. $m'_1 \neq m'_2$
2. $\{m'_1, m'_2\} \neq \{m_1, m_2\}$.
3. $\text{pad}(m'_1[n_1]) \oplus \text{pad}(m'_2[n_2]) = \text{pad}(m_1[n_1]) \oplus \text{pad}(m_2[n_2])$
4. The length of $m'_1[n_1]$ (resp., $m'_2[n_2]$) is less than l_B if and only if the length of $m_1[n_1]$ (resp., $m_2[n_2]$) is less than l_B.

where **pad** is the function that is used to pad the last block to length l_B. It is not hard to verify that two messages that satisfy the conditions above have identical authenticators when PMAC (resp., OMAC) is used for signing.

An algorithm that will forge on first collision works as follows. Given two message/authenticator pairs (m_1, τ) and (m_2, τ), we compute the messages m_1' and m_2'. Then, we seek in the set of known authentic pairs whether there is already a computed authenticator for m_1' or m_2'. If there is an already computed authenticator τ' for the message m_1' (resp., m_2'), then the algorithm outputs (m_2', τ') (resp., (m_1', τ')). The pair (m_2', τ') (resp., (m_1', τ')) is a forgery since (m_1, τ) and (m_2, τ) are the only pairs in P that collide. If there is no already computed authenticator for one of the messages m_1' and m_2', then the algorithm submits m_1' for signing and outputs (m_2', τ'), where τ' is the answer to the signing query. ■

3.4.4 Security based on PRFs

The result from the previous section can be generalized for any MAC scheme if we assume that the function family defined by the signing algorithm of the MAC scheme is pseudorandom. From the forgery or collision property, it follows that forging a signature in a 4C forest scheme implies constructing a forgery or collision for the underlying MAC scheme. Therefore, if one can forge signatures in a 4C forest scheme with significant probability, then one can distinguish the function family defined by the signing algorithm of the MAC scheme from a random function family as explained below in more details.

In our analysis, we assume that there is some limit l_M on the length of the message that can be submitted for signing to the underlying scheme. Furthermore, the tags computed by the scheme are of fixed length l_T. The randomness part of the tag has length l_R (l_R is zero if the scheme is not probabilistic) and the rest of the tag is $n_T = l_T - l_R$ bits long. Let Σ_{l_M} be the set of all non-empty strings over the alphabet $\{0,1\}$ whose length is at most l_M, let $\mathcal{R}$ be the family of all functions from the set $\{0,1\}^{l_R} \times \Sigma_{l_M}$ (Σ_{l_M} in the non-probabilistic case) to the set $\{0,1\}^{n_T}$, and let $\mathcal{F}$ be the family of functions defined by the signing algorithm of the MAC scheme. A statistical test is a Turing machine A with access to an oracle $\mathcal{O}$. The oracle is selected to be a random function from $\mathcal{F}$ or a random function from $\mathcal{R}$ according to a random bit b. The algorithm A outputs 0 or 1. The advantage of distinguishing the finite family $\mathcal{F}$ from the finite family $\mathcal{R}$ is defined as

$$\mathrm{Adv}_A(\mathcal{F}, \mathcal{R}) = \frac{1}{2}(E[A^F] - E[A^R])$$

where $E[A^F]$ (resp., $E[A^R]$) is the probability that A will output 1 when the oracles are selected to be random functions from $\mathcal{F}$ (resp., $\mathcal{R}$). We say that A $[t, q, \epsilon]$-breaks $\mathcal{F}$ if it runs in at most t

time, it makes no more than q queries, and it ϵ-distinguishes $\mathcal{F}$ from $\mathcal{R}$; that is $\mathrm{Adv}_A(\mathcal{F}, \mathcal{R}) \geq \epsilon$. The following theorem holds for a 4C forest scheme that uses a MAC scheme to compute the tags as described above.

Theorem 3.9 *Let E be an adversary that $[t, q_s, q_v, \epsilon]$-breaks a MAC based 4C forest scheme, let $q = L(q_s + q_v + 1)$, where L is the maximum number of internal nodes that can appear in a forest, let $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$, and let $\epsilon > p_R$. Then, there is a statistical test U that $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$-breaks the finite function family $\mathcal{F}$ defined by the signing algorithm of the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant.*

Proof. Given an adversary E that $[t, q_s, q_v, \epsilon]$-breaks a MAC based 4C forest scheme, we will construct the statistical test U in a following manner.

U's procedure is to run E and answer its oracle queries. In order to construct the answer to E's signing or verification query, U makes at most L queries to its oracle $\mathcal{O}$. U stores the result of each query in the memory and checks for collision in a following manner. Let m be the message corresponding to the query, let τ be the tag corresponding to the query and let (m, τ) be valid. If there is no message stored at address $\tau_0 + \tau$, where τ_0 is some constant integer, U stores m at that location. If there is a message stored at address $\tau_0 + \tau$ and the message is equal to m, U doesn't store anything and continues. If there is a message stored at address $\tau_0 + \tau$ and the message is not equal to m, U outputs 1 indicating that collision occurred and halts. If there are no two pairs that collide in the pool of known authentic pairs, E will finish and output its result. For each message/authenticator pair corresponding to the E's answer, U checks whether it is a forgery pair or it collides with some known authentic pair stored in the memory. If so, it outputs 1 and halts. Otherwise, it outputs 0 and halts. The number of queries that U submits to its oracle $\mathcal{O}$ is at most $L(q_s + q_v) + L = q$, and the time complexity of U is at most $t + cq$, where $c > 0$ is a small implementation dependent constant.

Now, consider the following problem. Given a randomly selected function F from $\mathcal{R}$ and a pool P of q authentic pairs, output a pair that is a forgery or collides with some pair in q, when the function F is used for authentication. Let (m, τ) be the output of an algorithm A that solves the problem. If (m, τ) is not in P and it is a valid pair, then the algorithm is successful. Since F is randomly selected from $\mathcal{R}$ the probability of success in this case is at most $p_f = 2^{-n_T}$ for any algorithm. Consider now the case when the algorithm outputs a pair (m, τ) that is in P. The probability of success is at most $p_c = 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$, where $\prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$ is the probability

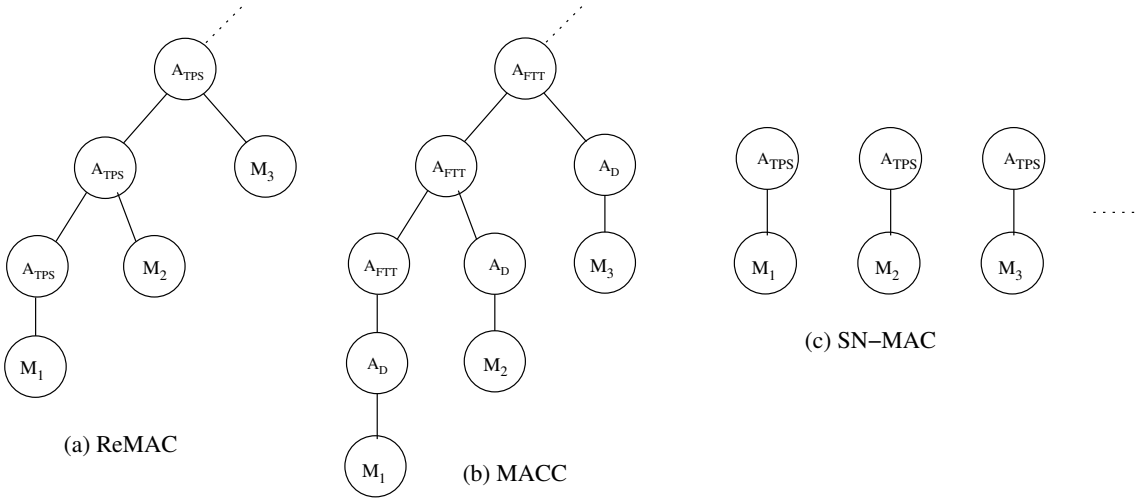

Figure 3.10: Practical stream authentication schemes

that there are no two pairs in P that collide. Hence, the probability that U will output 1, when the oracle is selected to be a random function from R is at most $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$. Therefore, the statistical test U that $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$-breaks the finite function family $\mathcal{F}$ defined by the signing algorithm of the underlying MAC scheme. ■

3.4.5 Security of ReMAC, MACC and SN-MAC

The security of SN-MAC, ReMAC and MACC follows trivially from the previous results.

A 4C forest representation of the schemes presented in Section 3.2 is given in Fig 3.10. ReMAC can be viewed as a 4C forest scheme where the forest consists only of one tree and all internal nodes are TPS nodes (see Fig 3.10.a). The following corollary establishes a relation between the security of a ReMAC scheme and the security of the underlying MAC scheme.

Corollary 3.10 *If there is an adversary that $[t, q_s, q_v, \epsilon]$-breaks ReMAC, then there is an adversary that $[t + cL(q_s + q_v + 1), L(q_s + q_v + 1), \epsilon]$-breaks the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant and L is the maximum number of chunks that can appear in a stream.*

The corollary follows from the Theorem 3.9 and Lemma 3.3. Since all internal nodes in the forest are TPS nodes, the adversary will always be able to find a forgery for the underlying scheme.

MACC scheme is a 4C tree scheme (or equivalently a forest scheme where the forest always has only one tree) such that there are no ITT or TPS nodes (see Fig 3.10.b). The security of the scheme follows from Theorem 3.9.

Corollary 3.11 *If there is an adversary that* $[t, q_s, q_v, \epsilon]$*-breaks the MACC scheme, then there is a statistical test that* $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$*-breaks the finite function family* $\mathcal{F}$ *defined by the signing algorithm of the underlying MAC scheme, where* $c > 0$ *is a small implementation dependent constant,* L *is twice the maximum number of chunks that can appear in a stream,* $q = L(q_s + q_v + 1)$ *and* $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$.

SN-MAC corresponds to a 4C forest scheme where the trees consist of only two nodes: a TPS root and a leaf (see Fig 3.10.c). Theorem 3.9 and Lemma 3.3 imply the following corollary about the security of SN-MAC.

Corollary 3.12 *If there is an adversary that* $[t, q_s, q_v, \epsilon]$*-breaks SN-MAC, then there is an adversary that* $[t + cL(q_s + q_v + 1), L(q_s + q_v + 1), \epsilon]$*-breaks the underlying MAC scheme, where* $c > 0$ *is a small implementation dependent constant and* L *is the maximum number of chunks that can appear in a stream.*

CHAPTER 4

Proven Secure Multicast Stream Authentication

The problems of multicast stream authentication and stream signing have been extensively studied in the past years. Gennaro and Rohatgi [43] have proposed a stream signing scheme based on a chain of one-time signatures. A similar scheme has been presented by Zhang [118] for authentication in routing protocols. Various schemes were proposed subsequently [28, 3, 116, 9, 99, 23, 111] culminating with the recent adoption of TESLA as an Internet standard (RFC4082). TESLA is also a basis for other internet drafts (e.g., [24]), and its in-depth security and efficiency analysis can be found in [88, 89, 90, 91].

In this chapter, we show that the suggested assumptions about the security of the building blocks of TESLA are not sufficient, and can lead to implementations that are not secure. We also provide sufficient security assumptions about the components of TESLA, and present secure implementations.

4.1 Insecure TESLA constructions from secure components

In this section, we show that the suggested assumptions about the building blocks of TESLA are not sufficient by providing examples of insecure TESLA constructions from components that satisfy those assumptions.

4.1.1 Permuted-input OMAC

In our analysis, we will use Permuted-input OMAC scheme to authenticate the packets of the stream. The scheme is depicted in Fig. 4.1. If the length of the message m is not greater than the block size n, then the authentication tag is computed as $\mathrm{OMAC}_K(m)$. Otherwise, the message m is rotated right by n bits to derive a new message m', and the authentication tag is computed as $\mathrm{OMAC}_K(m')$.

The unforgeability of POMAC trivially follows from the unforgeability of OMAC.

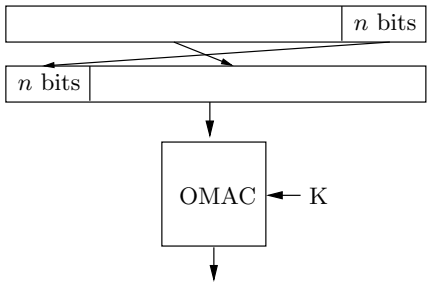

Figure 4.1: Permuted-input OMAC

Lemma 4.1 *Suppose that:*

- *h is a strongly collision resistant function (i.e., it is hard to find m_1 and $m_2 \neq m_1$ s.t. $h(m_1) = h(m_2)$), and*
- *$\{f_K\}_{K\in\{0,1\}^l}$ is a function family corresponding to an unforgeable MAC scheme.*

Then, the MAC scheme defined by the function family $\{f_K \circ h\}_{K\in\{0,1\}^l}$ is unforgeable too.

Corollary 4.2 *If the function family $\{E_K\}_{K\in\{0,1\}^l}$ defined by the underlying block cipher is a pseudorandom permutation family, then POMAC is unforgeable.*

Proof. Follows from Lemma 4.1 and the facts that the initial permutation in POMAC is a bijection (i.e., strongly collision resistant) and OMAC is unforgeable when the underlying block cipher is a pseudorandom permutation. ■

4.1.2 The case when F' is an identity mapping

In this section, we provide an example of an insecure TESLA construction from secure components in the case when the function F' is an identity mapping.

Suppose that the function family $\{E_K\}_{K\in\{0,1\}^n}$ is a target collision resistant pseudorandom permutation family whose members are defined on the set $\{0,1\}^n$. Note that the length of the key is equal to the block size n. AES-128 [41] is a possible candidate. Since $\{E_K\}_{K\in\{0,1\}^n}$ is a pseudorandom permutation family, it is also a pseudorandom function family (see Proposition 3.7.3 in [47]). We will use the pseudorandom permutation E_K to generate the authentication keys as illustrated in Figure 4.2. The key $K_{i-1} = E_K(0^n)$ is generated by encrypting 0 using the key K_i

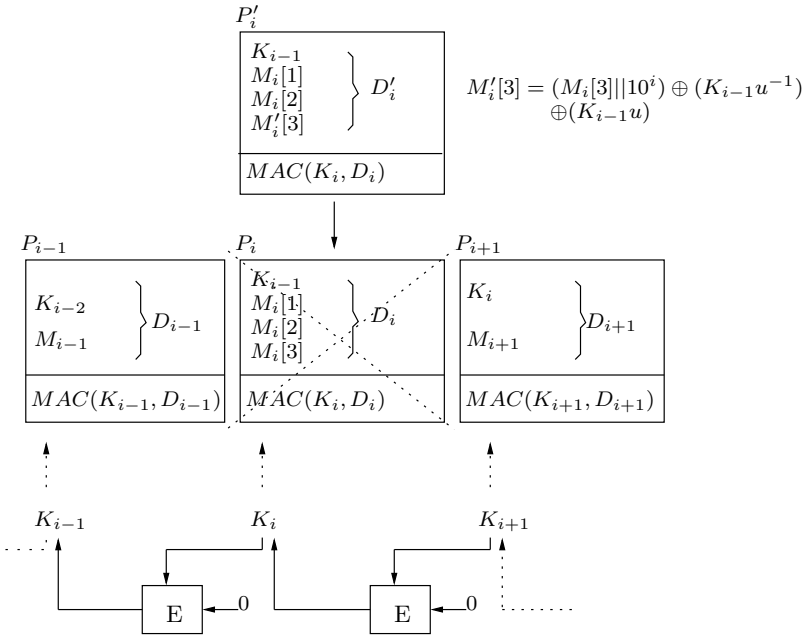

Figure 4.2: Insecure TESLA implementation. MACs are computed using POMAC.

as suggested in [88]. The MAC scheme that we use in our construction is POMAC. To encrypt the message blocks in POMAC, we use the pseudorandom permutation E_K.

The PRF and the MAC as defined above satisfy the security requirements of Theorem 1.5. However, the resulting stream authentication scheme is not secure. Figure 4.2 depicts an attack on our TESLA Scheme II example by replacing the packet P_i with a packet P'_i. Without loss of generality, we assume that the message M_i consists of three chunks $M_i[1], M_i[2]$ and $M_i[3]$. The length of $M_i[1]$ and $M_i[2]$ is equal to the block length n, and the length of $M_i[3]$ is less than the block length n. This implies that $M_i[3]$ is 10^i padded and XORed with $L \cdot u^{-1}$ when computing the MAC for P_i. The forged packet P'_i is constructed by replacing $M_i[3]$ with

$$M'_i[3] = (M_i[3]||10^i) \oplus (K_{i-1} \cdot u^{-1}) \oplus (K_{i-1} \cdot u),$$

where L is the encryption of zero and u is a public constant as in OMAC. Using the equations

$$(M_i[3]||10^i) \oplus (K_{i-1} \cdot u^{-1}) = M'_i[3] \oplus (K_{i-1} \cdot u)$$

and

$$L = E_{K_i}(0^n) = K_{i-1},$$

one can easily verify that

$$\mathrm{POMAC}(K_i, D_i) = \mathrm{POMAC}(K_i, D_i').$$

Note that all we need to compute P_i' is the key K_{i-1} and the message M_i. Since both the key K_{i-1} and the message M_i are disclosed in the packet P_i, we can compute P_i' before the key K_i is disclosed. Hence, we have succeeded in constructing a forgery for TESLA Scheme II.

The introduction of the POMAC scheme was motivated by the order of the message M_i and the key K_{i-1} within the packet P_i (Fig. 1.5). The initial permutation of POMAC swaps the message and the key so that the last block of D_i is a message block. In an implementation where the key K_{i-1} is the first block of D_i, the attack would work without modifying OMAC. Moreover, in the final version (TESLA Scheme IV), the format of the packets is $P_j = \langle M_j, i, K_{i-d}, MAC(K_i', M_j)\rangle$, where i is the interval during which the packet P_j was sent. Note that the MACs are computed over the messages M_j only, and the attack would work when OMAC instead of POMAC is used to compute the MACs. Hence, our analysis shows not only that the assumptions about the security properties of the building blocks of TESLA are not sufficient, but also that it is not unrealistic to expect that TESLA might be implemented insecurely.

4.1.3 The case when F' is implemented using a PRF

RFC4082 requires the function F' to be implemented using a pseudorandom function. The authentication key K_i' is computed as $K_i' = F'(K_i) = f'_{K_i}(1)$, where f' is a pseudorandom function. However, the new TESLA Scheme II still suffers from the flaw discussed in Section 4.1.2. Namely, we can view f' as a part of the key scheduling algorithm of the underlying block cipher. The function F of the insecure TESLA construction is now implemented as $F(K_i) = E_{f'_{K_i}(1)}(0)$ (see Figure 4.3). It is clear that $K_{i-1} = F(K_i)$ leaks the encryption of zero since $K_{i-1} = E_{K_i'}(0)$, and we can mount the same attack.

In addition to the old flaw, the modification of the scheme introduces a new one. Now, the commitment $F(K_i)$ might leak information about the authentication key K_i'. Consider the following "naive" implementation. The function F is implemented as $F(K_i) = f_{K_i}(0)$, where f is a target collision resistant pseudorandom function family. The function F' is implemented as $F'(K_i) = f'_{K_i}(1)$, where $f'_{K_i}(x) = f_{K_i}(x-1)$. One can easily show that f' is a pseudorandom function. It is not hard to verify that the commitment $F(K_i)$ discloses the authentication key K_i': $F(K_i) = f_{K_i}(0) = f_{K_i}(1-1) = f'_{K_i}(1) = K_i'$.

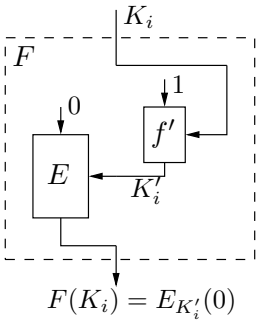

Figure 4.3: The function F leaks the encryption of zero $E_{K'_i}(0)$.

4.1.4 Cryptanalysis of the RFC4082 TESLA version

The analysis presented in Section 4.1.3 can be extended to the TESLA version described in RFC4082. The safe packet test only checks whether a packet authenticated using a key K_i was received before the disclosure of the key K_i. Hence, the adversary can delay the packet until the key K_{i-1} is disclosed, and then replace it with a forged one. The aforementioned security flaws cannot be patched by simply modifying the safe packet test so that the receiver checks whether the packet was received before the disclosure of the key value K_{i-1}. In this case, the adversary might be able to use $K_{i-2} = F(F(K_i))$ or some previous key value to mount an attack.

Furthermore, the format of the packets in the RFC4082 version is $P_j = \langle M_j, i, K_{i-d}, MAC(K'_i, M_j)\rangle$, where i is the time interval during which the packet P_j was sent. Observe that there are no sequence numbers that will specify the order of the packets that are sent within a given time interval. Hence, the adversary can rearrange the packets in this interval, and the receiver will accept them in a maliciously modified order. This is a violation of the unforgeability property.

Finally, we note that the original implementation of TESLA uses the MD5 hash function [98] in conjunction with HMAC construction [42] for the pseudo-random function and the MAC. The recent results on cryptanalysis of hash functions raised concern about the security of hash functions. In particular, an attack on MD5 and other hash functions was presented by Wang et al [113, 114] at Eurocrypt 2005. This implies that both the pseudo-random function and the MAC used in the TESLA implementation are not proven secure.

4.2 Sufficient assumptions about the components of TESLA

The attacks on the insecure implementations that were presented in Section 4.1 are based on the following observation. The security of the MAC scheme that is used to authenticate the packets is proven in a setting where the adversary has access to a signing oracle and a verifying oracle. In the case of TESLA, we have a different setting. Now, the adversary has access to an additional oracle that computes the commitment $F(K)$ to the secret key K which is used by the MAC scheme. The adversary can exploit the knowledge of $F(K)$ to construct a forgery.

It is clear from the discussion above that we need to make an additional assumption about the function F and the MAC scheme. Namely, the MAC scheme must remain secure even when the commitment of the secret key used by the MAC scheme is revealed.

Definition 4.1 *A MAC scheme is* known F-commitment unforgeable *if there is no efficient adversary that given a commitment $F(K)$ of the secret key that is in use can break the MAC scheme with non-negligible probability.*

We also make the following minor modification of TESLA Scheme II. Each time a stream is authenticated, the sender selects a unique number N_s (e.g., using a counter) which is securely communicated to the recipients. The number N_s is included as part of the authenticated data in each packet of the stream including the bootstrap packet. So, we assume that the format of the messages is $M_i = \langle N_s, i, C_i \rangle$, where C_i is the actual chunk of the stream[1].

The following theorem holds for the security of the slightly modified TESLA Scheme II.

Theorem 4.3 *Suppose that:*

1. *the digital signature scheme, which is used to bootstrap TESLA, is unforgeable,*
2. *the function $F(K) = f_K(0)$, where f is a pseudorandom function, is strongly collision resistant,*
3. *the MAC scheme, which is used to authenticate the chunks of the stream, is known F-commitment unforgeable, and*
4. *F' is an identity mapping.*

[1] To reduce the communication overhead one can communicate N_s only once, and then just use it to compute the signature and the MACs.

Then, TESLA Scheme II is a secure multicast stream authentication scheme.

Proof. Assume that the adversary can break the stream authentication scheme. In other words, the adversary in cooperation with some of the recipients can trick another recipient u to accept a forged packet of the stream as valid.

Let i be the smallest integer such that the contents D'_i is accepted as valid by the recipient u when the original contents D_i is different from D'_i. We consider the following events:

Event 1 If i is zero, then the adversary has managed to forge the bootstrap packet which was signed using a digital signature scheme.

Event 2 If i greater than zero and the key that u used to verify the validity of D'_i is equal to the original key K_i, then the adversary has managed to produce a forgery for the message authentication scheme due to the uniqueness of $\langle N_s, i\rangle$.

Event 3 If i is greater than zero and the key K_i^f that u used to verify the validity of D'_i is different than the original key K_i, then the adversary has managed to find a collision for the function F. Let $K_i^f, K_{i-1}^f = F(K_i^f), \ldots$ be a key chain derived from K_i^f, and let $K_i, K_{i-1} = F(K_i), \ldots$ be a key chain derived from K_i. The user u verified the validity of the key value K_i^f by checking whether $F^l(K_i^f)$ is equal to some previously authenticated key value K_{i-l}^f. Since i is the smallest index of a packet whose contents D'_i is different from the original contents D_i, the received key value K_{i-l}^f must be equal to the original key value $K_{i-l} = F^l(K_i)$. Hence, there is an index $i - l \leq j < i$ s.t. $K_{j+1} \neq K'_{j+1}$ and $F(K_{j+1}) = K_j = K'_j = F(K'_{j+1})$.

Given an efficient adversary A_{SA} that breaks the stream authentication scheme with significant probability, we will construct an adversary A_{S} for the signature scheme, an adversary A_{MAC} for the MAC scheme and an adversary A_{F} for the function F, and show that at least one of these adversaries has significant success probability. All three adversaries simulate the network using sets of read and write tapes for the users and for the adversary. They differ in the following aspects:

1. The adversary for the signature scheme answers the stream signing queries by randomly selecting initial key values, computing the key chains and using the signing oracle for the bootstrap packets. Whenever A_{SA} manages to forge a bootstrap packet, A_{S} outputs the forged message/signature pair. Otherwise, it outputs a randomly selected message/signature pair.

2. The adversary for the MAC scheme guesses which stream will be forged and what will be the smallest index i of a forged packet within the stream. If the guess is that the stream will not be forged, then A_{MAC} answers the stream signing query by randomly selecting the initial key value. Otherwise, the adversary uses the given value $K_{i-1} = F(K_i)$ to derive the keys that will be used to authenticate the packets $P_1, \ldots P_{i-1}$, and computes the MAC for the packet P_i by submitting a query to the (MAC) signing oracle. If the adversary for the stream scheme manages to forge the i-th packet, then the adversary for the MAC scheme outputs the forged message/MAC pair. Otherwise, it outputs a randomly selected message/MAC pair.

3. The adversary for the function F answers the stream signing queries by randomly selecting initial key values, computing the key chains and using a private key when signing the bootstrap packets. In the case when Event 3 occurs, A_{F} finds and outputs a pair of key values that collide. Otherwise, it outputs two randomly selected key values.

It is easy to show that if the probabilities of Event 1 and Event 3 are significant, then the success probabilities of the corresponding adversaries A_{S} and A_{F} are significant too. To derive a relation between the probability of Event 2 and A_{MAC}, we need the following Lemma.

Lemma 4.4 *If f is a pseudorandom function, then there is no efficient algorithm that can distinguish between a random key value and the key value $F^l(K)$ derived from a secret random key K by $l \geq 1$ iterations of the function F.*

Proof. We can prove the Lemma by induction. If there is an algorithm that can tell apart $F(K) = f_K(0)$ from a random key value, then we can construct an algorithm that can distinguish between the function family $\{f_K\}$ defined by f and the random function family. Now, assume that there is no algorithm that can tell apart between the key $K_{l-1} = F^{l-1}(K)$ and a random key value. Since the function f and the key K_{l-1} are pseudorandom, the key $K_l = f_{K_{l-1}}(0)$ will be indistinguishable from a random key too. ■

Assume that n_s and L are the maximum number of streams and the maximum number of packets within a single stream respectively. The probability that A_{MAC} will guess the forged stream and the index i of the first forged packet within the stream is $\frac{1}{n_s L}$. According to Lemma 4.4, there is no efficient algorithm that can distinguish with significant probability between the secret key K_i used by the MAC scheme and a key that is derived from some initial key value by $l - i$ iterations

of the function F. Hence, if the probability ϵ of Event 2 is significant, then the success probability of A_{MAC} will be approximately $\frac{\epsilon}{n_s L}$. ■

The requirement for strong collision resistance of the function F can be slightly weakened. Assuming that there is a bound on the number of packets within a stream, it is not hard to show that TESLA Scheme II is secure when the function F is collision resistant in the following sense: Given a randomly selected value K and a bound $L \geq 1$, it is hard to find K' and a positive integer $l \leq L$ such that $F^l(K) = F(K')$ and $F^{l-1}(K) \neq K'$. A function that satisfies the aforementioned property is said to be *bounded iteration collision resistant*. One can easily prove that strong collision resistance implies bounded iteration collision resistance, and that bounded iteration collision resistance implies weak (second pre-image) collision resistance.

4.3 Secure TESLA implementation via a CKDA-secure pseudorandom permutation

In this section, we propose an implementation that uses block ciphers to realize the different components of TESLA.

4.3.1 A related-key model of a block cipher

When used as components of more complex constructions, block ciphers are usually modeled as pseudorandom permutations. However, when cryptanalyzed, block ciphers are not considered secure unless they are resistant to related-key attacks [13]. Here, we are going to use a model of the second setting where the adversary can query oracles that use keys whose difference was chosen by the adversary.

We are going to define a CKDA-secure (i.e., secure against Chosen Key Difference Attacks) pseudorandom permutation family as a pseudorandom permutation family such that one cannot tell apart a pair of permutations randomly selected from the family and a pair of permutations from the family whose index (key) difference is selected by the adversary. A chosen key difference test is a Turing machine A with access to four oracles $\mathcal{E}_1$, $\mathcal{D}_1$, $\mathcal{E}_2$ and $\mathcal{D}_2$. A selects a non-zero l-bit string c. The oracle $\mathcal{E}_1$ is selected to be a random permutation E_K from the permutation family $\{E_K\}_{K \in \{0,1\}^l}$, and the oracle $\mathcal{D}_1$ is selected to be its inverse. According to a secret random bit b, the oracle $\mathcal{E}_2$ is selected to be either the permutation $E_{K \oplus c}$ or a random permutation $E_{K \oplus r}$, where c is the public non-zero constant and r is a random bit string of length l. The oracle $\mathcal{D}_2$ computes

the inverse of $\mathcal{E}_2$. The algorithm A outputs 0 or 1. The advantage of the CKD test is defined as

$$\mathrm{Adv}_A((E_K, E_{K\oplus c}), (E_K, E_{K\oplus r})) = \frac{1}{2}(E[A^C] - E[A^R])$$

where $E[A^C]$ (resp., $E[A^R]$) is the probability that A will output 1 when the difference between the secret keys is a chosen non-zero constant (resp., random l-bit string).

Definition 4.2 *The pseudorandom permutation family* $\{E_K\}_{K\in\{0,1\}^l}$ *is a* $[t, q, \epsilon]$-CKDA-secure pseudorandom permutation family *if there is no CKD test that runs in at most* t *time, sends at most* q *queries to the oracles and has at least* ϵ *advantage.*

In the model described here, the known relation between the keys is their difference. This corresponds to a block cipher resistant to related-key differential cryptanalysis [66, 60]. Other variants and generalizations are also possible. Another possible definition of CKDA-secure PRP family is as a permutation family s.t. a pair of related members of the family is indistinguishable from a pair of random permutations. It is not hard to verify that this definition is equivalent to the one that we use here.

Finally, we must note that although we will use CKDA-secure pseudorandom permutations to provide known commitment unforgeability, there are other possible applications of this model. Consider a proven secure encryption scheme and a proven secure message authentication scheme. Instead of using two different randomly selected secret keys for these schemes, we can use one randomly selected key to derive two related keys that will be used by the schemes. In most of the cases, using the related-key model, it would be trivial to prove that the schemes will remain secure. So, we have secure schemes that require 50% less randomness and key storage size.

4.3.2 A candidate implementation of TESLA

The following theorem provides a function F and a MAC scheme such that the MAC scheme is known F-commitment unforgeable.

Theorem 4.5 *Let the function family* $\{E_K\}_{K\in\{0,1\}^n}$ *corresponding to the block cipher used by OMAC be CKDA-secure PRP family. Let* $F : \{0,1\}^n \to \{0,1\}^n$ *be defined as* $F(K) = E_{K\oplus c}(0)$, *where* $c = 0^{n-1}1$. *Then, OMAC is a known* F*-commitment unforgeable MAC scheme.*

Proof. Let A_1 be an adversary who given a commitment $F(K')$ to some randomly selected key K' can break OMAC with probability ϵ. Such an adversary A_1 can be easily converted into

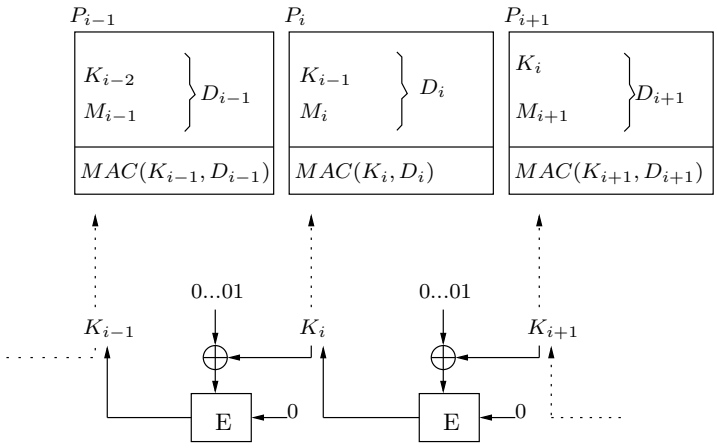

Figure 4.4: TESLA implementation using a block cipher resistant to related-key cryptanalysis

an adversary A_2 that can break OMAC with the same probability. In particular, A_2 can randomly select the key value K' and submit the commitment $F(K')$ to A_1. A_1's output will be A_2's output. Since OMAC is unforgeable, there is no adversary that can break OMAC with significant probability given a commitment to a randomly select key.

Now, assume that there is an adversary A_3 that can break OMAC given the commitment $F(K)$ to the secret key K that is in use. We can construct a CKD test as follows. We run the adversary A_3 and answer its queries by querying the encryption oracles $\mathcal{E}_1$ and $\mathcal{E}_2$. If A_3 manages to produce a forgery we output 1, otherwise we output 0. Obviously, the advantage of the CKD test will be significant since the probability $E[A^C]$ is significant (OMAC is not known F-commitment unforgeable) and the probability $E[A^R]$ is small (OMAC is unforgeable). ■

The implementation that we propose here is depicted in Figure 4.4. It is similar to the insecure implementation from Figure 4.2. However, the key value K_{i-1} is derived by encrypting zero using the key $K_i \oplus 0^{n-1}1$ instead of the key K_i. A similar secure variant of the insecure implementation can be obtained by using a function F' that derives the key K'_i by flipping the last bit of K_i instead of using an identity map.

4.4 Secure TESLA implementation via erasure-tolerant authentication codes

In this section, we present a new erasure-tolerant stream authentication scheme constructed using unconditionally secure erasure-tolerant authentication codes (η-codes).

Most of the multicast stream authentication schemes are based on the concept depicted in Figure 4.5.a (see [88]). To authenticate the packet (chunk) P_i of the stream, the sender first commits to the key value K_i by sending $H(K_i)$ in the packet P_{i-1}. The key K_i is only known to the sender, and it is used to compute a MAC on the packet P_i. After all recipients have received the packet P_i, the sender discloses the key value K_i in the packet P_{i+1}. The recipients verify whether the received key value corresponds to the commitment and whether the MAC of the packet P_i computed using the received key value corresponds to the received MAC value. If both verifications are successful, the packet P_i is accepted as authentic. Note that P_i contains the commitment to the next key value K_{i+1}. To bootstrap the scheme, the first packet, which contains the commitment $H(K_1)$ to the first symmetric key K_1, is signed using a digital signature scheme (e.g., RSA).

Our scheme is derived by modifying the basic scheme depicted in Figure 4.5.a. We divide the stream into groups, each group i being a sequence of b messages $M_{i,1}, \ldots, M_{i,b}$. A unique sequence number $\langle N_s.i.j \rangle$ is assigned to each message $M_{i,j}$, where $\langle N_s.i.j \rangle$ is a concatenation of the binary representations of a unique stream number N_s, the group number i and the position of the message in the group j. A commitment $H(S_{i+1})$ to a key string S_{i+1} and a key string S_{i-1} are included in $t+1$ packets of the group[2]. Since at most t erasures are allowed, at least one commitment copy $H(S_{i+1})$ and at least one key string copy S_{i-1} will get to the receiver. Finally, we compute and include v authentication tags in the sequence of b packets. The authentication tags are computed using an unconditionally secure authentication codes as in the construction from cover-free families described in Section 2.6.2. The keys that are used to compute the tags are extracted from S_i, which is revealed in the next group of packets. If the number of tags v is large, then S_i will be a significant communication overhead. One possible solution to this problem is to use S_i as a short seed of a pseudo-random generator that will generate the secret keys of the unconditionally secure authentication codes.

In our simple example (Figure 4.5.b), the stream is divided into groups of 16 messages. Since only one erasure is allowed, the commitment $H(S_{i+1})$ and the seed S_{i-1} are included in only two

[2]In general, we can use an erasure code to encode the commitment and the key string instead of sending multiple copies. This further optimizes the bandwidth.

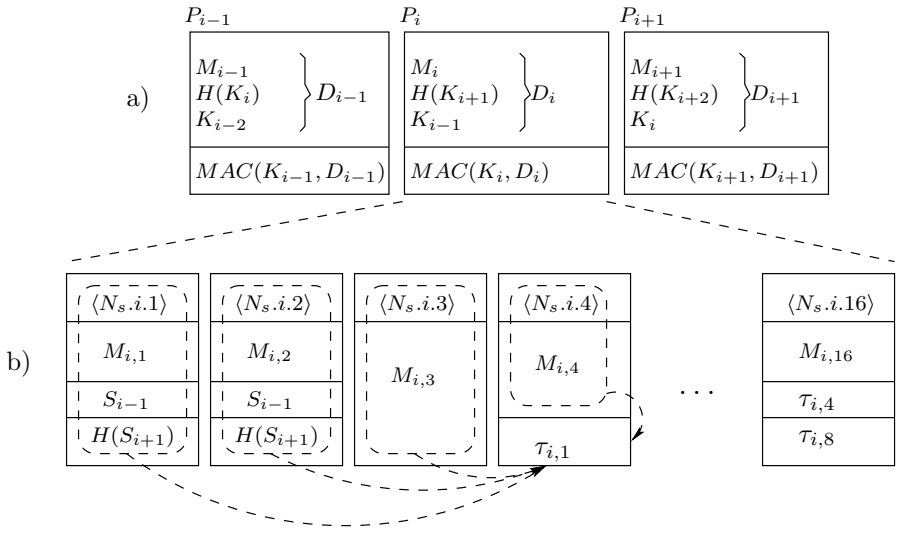

Figure 4.5: Multicast stream authentication: a) The basic scheme, b) Using $(1,1)$-CFF

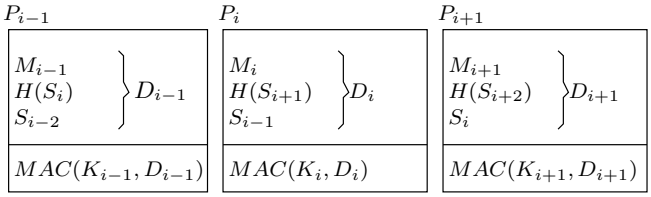

Figure 4.6: A variant of the basic stream authentication scheme.

packets. The authentication tags are computed using a $(1,1)$-cover-free family as in Figure 2.1. The only difference is that the tags depend on the whole content of the packets, not just on the messages $M_{i,j}$.

In order to analyze the security of the scheme, we are going to consider a variant of the basic scheme depicted in Figure 4.6. Instead of committing to the key K_i, the sender commits to a string $S_i = K_i||r_i$, which is a concatenation of the key K_i and a random string r_i. A security proof of the erasure-tolerant stream authentication scheme depicted in Figure 4.4 can be easily obtained by extending the following result about the security of the modified basic scheme.

Theorem 4.6 *Suppose that:*

- $H(S_i)$ *provides a perfect concealing (i.e., even a computationally unbounded adversary cannot*

do better than randomly guess the value of K_i) and computationally secure binding (i.e., there is no efficient adversary that can find a key $K'_i \neq K_i$ and a random value r'_i s.t. $H(K'_i||r'_i) = H(K_i||r_i)$).

- *the digital signature scheme that is used to bootstrap the scheme (i.e., sign the first packet with a commitment $H(S_1)$ to the first key K_1) is unforgeable, and*
- *the message authentication scheme is unconditionally secure.*

Then, the variant of the basic scheme depicted in Figure 1.4 is computationally secure.

Proof. Assume that the adversary can break the stream authentication scheme. In other words, the adversary in cooperation with some of the recipients can trick another recipient u to accept a forged packet of the stream as valid. Let i be the smallest integer such that the content D'_i is accepted as valid by the recipient u when the original content D_i is different from D'_i. If i is zero, then the adversary has managed to forge the bootstrap packet which was signed using a digital signature scheme. This is in contradiction with our assumption that the digital signature scheme is unforgeable. Let i be greater than zero. There are two possible cases: (i) the key that u used to verify the validity of D'_i is equal to the original key K_i or (ii) the key that u used to verify the validity of D'_i is different from the original key K_i. In the first case, the adversary has managed to produce a forgery for the message authentication scheme. Since the message authentication scheme is unconditionally secure and the commitment $H(S_i)$ does not leak any information about the key K_i, the probability of success in this case is very small even for a computationally unbounded adversary. In the second case, when the key K'_i used by u is different from the original key K_i, the string S'_i received by u is different from the original string S_i sent by the sender. However, the corresponding commitments $H(S_i)$ and $H(S'_i)$ must be equal since i is the smallest number such that the content D'_i is accepted as valid by the recipient u when the original content D_i sent by the sender is different from D'_i. This is in contradiction with our binding assumption. ■

CHAPTER 5

Related-Key Differential Cryptanalysis of AES

In the previous chapter, we proposed a secure TESLA implementation that used block cipher to realize the different components of TESLA. The security of the implementation relies on the assumption that the block cipher is a CKDA-secure pseudorandom permutation. If this assumption is not true, then the security proof collapses. Hence, in this chapter, we investigate the resistance of AES to related-key differential attacks[1].

5.1 Related-key differential attacks

Differential cryptanalysis exploits the propagation of the differences when a pair of distinct plaintexts is submitted for encryption under the same key. Related-key differential cryptanalysis exploits the properties of the difference propagation when the plaintexts x_1 and x_2, which can be equal, are submitted for encryption under distinct keys k_1 and k_2 correspondingly.

An r-round related-key differential is a triple (α, β, δ), where α is the difference of the inputs at the input of the block encryption algorithm, β is the difference of the outputs of the r-th round and δ is the difference of the keys. The probability of r-round related-key differential is the probability that the difference of the outputs of the r-th round will be β, when the input difference is α, the key difference is δ, and the plaintext x_1 and the key k_1 are selected uniformly at random.
The general description of the related-key differential attacks is similar the one described in Section 1.1.1 for ordinary differential attacks:

- Find highly probable $R-1$-round related-key differential, where R is the number of rounds of the block cipher.
- Select randomly x_1 and submit it for encryption under key k_1 to obtain ciphertext y_1. Compute $x_2 = x_1 + \alpha$ and submit it for encryption under key $k_2 = k_1 + \delta$ to obtain the

[1] This chapter is based on [60].

ciphertext y_2.

- Find all possible last round key pairs (k_1^R, k_2^R) such that the difference between $d_{k_1^R}(y_1)$ and $d_{k_2^R}(y_2)$ is β, where $d_k(y)$ is the output of the first round of the decryption algorithm for input y and round key k. Add one to each counter that corresponds to one of the previously computed key pairs.

- Repeat previous two steps until one or more last round key pairs are counted significantly more than the others. Check these keys if they are the right keys.

Let K be the number of possible last round key pairs (k_1^R, k_2^R) and let l be the average number of suggested key pairs in the third step. Furthermore, let $p_{max} >> 2^{-m}$, where m is the block length and p_{max} is the probability of the related-key differential found in step 1 and let $N = c/p_{max}$ be the number of repetitions of steps two and three. Then the wrong key pairs will be counted lN/K times on average and the right key pair will be counted about c times on average. If $l \times c < K \times p_{max}$, then the wrong key pairs will be counted less then once on average. The order of the number of required plaintext/ciphertext pairs is $1/p_{max}$.

The related-key differential attack does not have to follow the pattern described above. In general, we will refer to any attack that exploits a related-key differentials as related-key differential attack.

5.2 Related-Key Differential Attacks on AES-192

In this section, we describe some attacks on reduced 192-bit key variants of AES. Description of the algorithm can be found in [41]. We will use the following notation: $x_i^I, x_i^S, x_i^P, x_i^M$ and x_i^O denote the input of the round i, the output after SubBytes, the output after ShiftRows, the output after MixColumns and the output after AddRoundKey transformation, correspondingly; k_i denotes the round i key and we will use $a_{i,j}, i, j \in \{0, 1, 2, 3\}$ to denote the byte j of the 32-bit word (column) i of a. For example, $x_{3,0,2}^P$ denotes the third byte of the first column of the output after the ShiftRows transformation in the round 3 (the initial key addition of the algorithm is considered as round 0). The encryption round can be transformed into equivalent one that uses key addition before the MixColumns. The round keys used in the equivalent encryption round will be denoted by z_i. Finally, we assume that the last round of the analyzed variants is a FinalRound.

Table 5.1: Propagation of the key difference $(0000)(0000)(0\Delta 00)(0\Delta 00)(0000)(0000)$

j	Δk_j
0	$(0000)(0000)(0\Delta 00)(0\Delta 00)$
1	$(0000)(0000)(0000)(0000)$
2	$(0\Delta 00)(0000)(0000)(0000)$
3	$(0000)(0000)(0\Delta 00)(0\Delta 00)$
4	$(0\Delta 00)(0\Delta 00)(\Delta_1 000)(\Delta_1 000)$
5	$(\Delta_1 \Delta 00)(\Delta_1 000)(\Delta_1 \Delta 00)(\Delta_1 000)$
6	$(\Delta_1 00\Delta_2)(000\Delta_2)(\Delta_1 \Delta 0\Delta_2)(0\Delta 0\Delta_2)$

Table 5.2: Possible propagation of the plaintext difference

j	Δx_j^I
0	$(0000)(0000)(0\Delta 00)(0\Delta 00)$
1	$(0000)(0000)(0000)(0000)$
2	$(0000)(0000)(0000)(0000)$
3	$(0\Delta 00)(0000)(0000)(0000)$
4	$(0000)(0000)(0\Delta 00)(\text{`}03' \cdot \Delta' \| 0\Delta'\Delta')$
	$\Delta x_4^S = (0000)(0000)(0\Delta 00)(\Delta'' 0\Delta\Delta)$
5	$(\Delta 0 \| \text{`}03' \cdot \Delta \| \text{`}02' \cdot \Delta)(\text{`}02' \cdot \Delta \| 0 \| \text{`}03' \cdot \Delta \| 0)$
	$(\Delta_1 000)(\text{`}02' \cdot \Delta'' \oplus \Delta_1 \| \Delta''\Delta'' \| \text{`}03' \cdot \Delta'')$

5.2.1 The Basic Attack

Differential cryptanalysis attacks are based on difference propagations over all but few rounds that have large enough prop ratio. For Rijndael, it is proven that any 4-round differential trail has at least 25 active bytes, that is there are no 4-round differential trails with predicted prop ratio above 2^{-150} [30, 31]. The idea of the attack described here is to use the round key differences in order to cancel the differences that exist before the key addition and reduce the number of active bytes.

The propagation of the key difference $(0000)(0000)(0\Delta 00)(0\Delta 00)(0000)(0000)$ is depicted in Table 5.1. If we submit two plaintexts x and x' for encryption under the keys k and $k' = k \oplus \Delta k$ correspondingly, such that $\Delta x = (0000)(0000)(0\Delta 00)(0\Delta 00)$, then a possible propagation of the difference is the one shown in Table 5.2.

The difference Δ' is selected to satisfy the relation $\text{`}02' \cdot \Delta' = \Delta$. In addition, the differences $\Delta, \Delta'$ and $\Delta''$ should be selected so that the probabilities $P_S(\Delta \to \Delta), P_S(\Delta \to \Delta'), P_S(\Delta' \to \Delta)$ and $P_S(\text{`}03' \cdot \Delta' \to \Delta'')$ are greater than zero, where $P_S(a \to b)$ is the probability that the output difference of the S-box will be b when the input difference is a. When the previous conditions

are satisfied, the 5-round related-key differential trail from Table 5.2 has 15 active bytes and its probability is $2^{-7\times 15} = 2^{-105}$ in the worst case. If we use 2^{106} plaintext/ciphertext pairs in a related-key differential attack on the six round variant, then the right key will be counted at least twice on average. The time complexity of the attack is about 2^{112} encryptions.

5.2.2 Improving the Basic Attack: Truncated Differentials

Let us consider the same plaintext and key differences as in the previous subsection. The probability that $\Delta x^I_{5,2} = \Delta_1 000$ is the same as the probability $P_S(\Delta \to \Delta')$ and it is 2^{-7} instead of 2^{-32}, which is the probability when the difference $\Delta x^I_{5,2}$ is uniformly distributed. This highly probable truncated differential [67] is exploited in the attack on six round version of AES. The attack is described below.

We assign counter to every 10-tuple

$$(\Delta_1, \Delta_2, k_{6,0,0}, k_{6,0,1}, k_{6,0,2}, k_{6,0,3}, k_{6,1,1}, k_{6,2,0}, k_{6,3,6}, z_{5,2,0})$$

that is possible for a given Δ. The attack is as follows:

- Select randomly a plaintext pair (x, x') such that the plaintext difference is

  $$(0000)(0000)(0\Delta 00)(0\Delta 00)$$

  and the ciphertext difference is $\Delta y = (****)(0*0\Delta_2)(*\Delta 0\Delta_2)(0\Delta 0*)$, where '*' means any value, $y = E_k(x), y' = E_{k\oplus\delta}(x')$ and $\delta = (0000)(0000)(0\Delta 00)(0\Delta 00)(0000)(0000)$.

- For every possible 10-tuple, check whether $\Delta x^I_{5,2} = \Delta_1 000$. The value of $\Delta x^I_{5,2,0}$ can be determined from the ciphertext pair using the key differences and the five bytes of the key $k_{6,0,2}, k_{6,1,1}, k_{6,2,0}, k_{6,3,3}$ and $z_{5,2,0}$. Further, the difference $\Delta x^P_{6,i,j}$ is zero for all i and j such that $k_{6,i,j}$ is unknown. Hence, one can compute the differences Δx^O_5, and therefore,the value Δx^S_5 can also be computed due to the linearity of the MixColumns transformation. Now, it is easy to check whether the particular difference before the SubBytes transformation of the round 5 is zero. If $\Delta x^I_{5,2} = \Delta_1 000$, then add one to the counter that corresponds to the particular 10-tuple.

- Repeat the previous two steps until one or more of the 10-tuples are counted significantly more than the others. Take this values as possible values of the specified bytes of the key.

Table 5.3: Propagation of the key difference (0000)(0000)(Δ000)(Δ000)(0000)(0000)

j	Δk_j
0	(0000)(0000)(Δ000)(Δ000)
1	(0000)(0000)(0000)(0000)
2	(Δ000)(0000)(0000)(0000)
3	(0000)(0000)(Δ000)(Δ000)
4	(Δ000)(Δ000)(000Δ_1)(000Δ_1)
5	(Δ00Δ_1)(000Δ_1)(Δ00Δ_1)(000Δ_1)
6	(00$\Delta_2\Delta_1$)(00$\Delta_2$0)(Δ0$\Delta_2\Delta_1$)(Δ0$\Delta_2$0)
7	(00$\Delta_2\Delta_1$)(00$\Delta_2$0)(0$\Delta_3\Delta_2\Delta_1$)(0$\Delta_3$0Δ_1)

If we repeat the first two steps 2^8 times, then the right 10-tuple will be counted twice on average, while the wrong 10-tuples will be counted 2^{-24} times assuming that when we use wrong key values the probability distribution of $\Delta x^I_{5,2}$ is uniform. Further, the probability that the output difference Δy will be $(***)(0*0\Delta_2)(*\Delta0\Delta_2)(0\Delta0*)$, when the plaintext difference is Δx = (0000)(0000)(0Δ00) (0Δ00), is $2^{-9\times8} = 2^{-72}$. Therefore, the number of plaintext pairs required for the attack is about $2^{72} \times 2^8 = 2^{80}$. There are at most 2^{14} possible values of (Δ_1, Δ_2) for a given Δ. Hence, the complexity of the attack is about $2^8 \times 2^{14} \times 2^{8\times8} = 2^{86}$ encryptions. The previously described attack is used to determine eight bytes of the key. It is not difficult to find the rest of the key using similar methods.

5.2.3 Impossible Related-Key Differentials Attacks

An impossible differential attack against Rijndael reduced to five rounds was proposed by Biham and Keller [15]. Later, this attack was extended to six rounds [27]. In this section, we describe related-key impossible differentials attacks on 192-bit key variant reduced to seven and eight rounds.

The attack exploits a similar weakness in the key schedule as the previous attacks. Namely, if the key difference is (0000)(0000)(Δ000)(Δ000)(0000)(0000), then this difference is propagated during the key generation as depicted in Table 5.3. We can see that the Round 1 key difference is zero and the Round 2 keys differ in only one byte. If we submit two plaintexts x and x' for encryption, such that Δx = (0000)(0000)(Δ000)(Δ000), then Δx^I_1 is zero, and so is $\Delta x^O_1 = \Delta x^I_2$. Because of the Round 2 key difference, the inputs of the third round will differ in only one byte $x^I_{3,0,0}$. Due to the MixColumn transformation and the Round 3 key difference, the inputs of the Round 4 will differ in six bytes $x^I_{4,0,1}, x^I_{4,0,2}, x^I_{4,0,3}, x^I_{4,1,1}, x^I_{4,1,2}$, and $x^I_{4,1,3}$. Hence, $\Delta x^M_{5,3} \neq 0000$ and

$\Delta x^O_{5,3} \neq 000\Delta_1$.

The aforementioned fact can be used to find values of seven bytes of the last round key. Given Δ, for every possible 10-tuple

$$(\Delta_1, \Delta_2, k_{7,0,0}, k_{7,1,0}, k_{7,1,1}, k_{7,1,2}, k_{7,1,3}, k_{7,2,2}, k_{7,3,1}, z_{6,0,3})$$

do the following:

- Compute Δ_3 using $k_{7,1,2}$ and Δ_2.
- For a plaintext pair (x, x') such that $\Delta x = (0000)(0000)(\Delta 000)(\Delta 000)$ and the ciphertext difference is $\Delta y = (*0\Delta_2\Delta_1)(****)(0\Delta_3 * \Delta_1)(0 * 0\Delta_1)$, where $y = E_k(x), y' = E_{k\oplus\delta}(x')$ and $\delta = (0000)(0000)(\Delta 000)(\Delta 000)(0000)\ (0000)$, check whether $\Delta x^O_{5,3} = 000\Delta_1$. The value of $\Delta x^O_{5,3,3}$ can be determined from the ciphertext pair using the key differences and the five bytes of the key $k_{7,0,0}, k_{7,1,3}, k_{7,2,2}, k_{7,3,1}$ and $z_{6,0,3}$. Further, the difference $\Delta x^P_{7,i,j}$ is zero for all i and j such that $k_{7,i,j}$ is unknown. Hence, one can compute the differences Δx^O_6 and therefore Δx^S_6 due to the linearity of the MixColumns transformation. Once the value of Δx^S_6 is determined, it is not difficult to check whether $\Delta x^O_{5,3} = 000\Delta_1$. If $\Delta x^O_{5,3} = 000\Delta_1$, then mark the current 10-tuple as wrong.
- Repeat the previous step until the 10-tuple is marked as wrong or the maximum of 2^{38} tried plaintext pairs is reached.

The probability that the ciphertext difference will be $\Delta y = (*0\Delta_2\Delta_1)(****)(0\Delta_3 * \Delta_1)(0 * 0\Delta_1)$, when the plaintext x is randomly selected, is $2^{-9\times 8} = 2^{-72}$. Hence, the number of plaintext pairs required to obtain 2^{38} plaintext pairs with the desired property is about 2^{110}. Given Δ, the number of possible values of (Δ_1, Δ_2) is less than 2^{14}. Thus, the complexity of finding the possible 10-tuples is of order $2^{38} \times 2^{14} \times 2^{8\times 8} = 2^{116}$ encryptions. The probability that particular wrong 10-tuple will be marked as wrong using only one pair of plaintexts is 2^{-32}. The number of wrong 10-tuples that are not marked as wrong after applying the procedure 2^{38} times is on average $2^{14} \times 2^{64} \times (1 - 2^{-32})^{2^{38}} \approx 2^{78} \times e^{-2^6} \approx 2^{-14}$ i.e. most of the time there will be no wrong keys that are not marked as wrong. The previous procedure is used to find eight bytes of the key. The rest of the key can be determined using similar techniques with complexity which is negligible compared to the complexity of the overall attack.

The attack can be extended to eight rounds. We will use the same plaintext and key differences, but we will use the fact that $\Delta x^M_{5,1} \neq 0000$ and $\Delta x^O_{5,1} \neq 000\Delta_1$, which can be proved to be true by similar arguments as in the previous case.

Given Δ, for every possible 3-tuple

$$(k_8, z_{7,2,0}, z_{7,3,3})$$

do the following:

- Compute: $k_{7,0}, k_{7,1}, k_{6,3}, k_{5,2}, k_{5,3}, k_{4,1}$ and $z_{8,0,3}$ using k_8; Δ_1 using $k_{4,1}$ and Δ; Δ_2 using $k_{5,3}$ and Δ_1; Δ_3 using $k_{7,1}$ and Δ_2; and finally, $z_{5,2,3}$ using $z_{7,3,3}$ and $z_{8,0,3}$.

- For a plaintext pair (x, x') such that $\Delta x = (0000)(0000)(\Delta 000)(\Delta 000)$ and the difference $\Delta^{a,P}_7$ is $(****)(****)(*000)(000*)$, check whether $\Delta x^O_{5,1} = 000\Delta_1$. The difference $\Delta^{a,P}_7$ is the difference after the ShiftRows transformation of the Round 7 computed using the assumed value k_8 from the ciphertext pair obtained when x is submitted for encryption under k and x' is submitted for encryption under $k \oplus \delta$. The value of $\Delta x^O_{5,1,3}$ can be determined from the ciphertext pair using the key differences, $k_8, k_{7,0}, k_{7,1}, z_{7,2,0}, z_{7,3,3}$ and $z_{5,2,3}$.Further, the difference $\Delta x^P_{7,i,j}$ is zero for all i and j such that $z_{7,i,j}$ can not be computed. Hence, one can compute the differences Δx^O_6, and therefore Δx^S_6 also due to the linearity of the MixColumns transformation. Now, it is easy to check whether the particular difference before the SubBytes transformation of the Round 6 is zero. If $\Delta x^O_{5,1} = 000\Delta_1$, then mark the current 3-tuple as wrong.

- Repeat the previous step until the 3-tuple is marked as wrong or the maximum of 2^{39} tried plaintext pairs is reached.

The probability that the difference $\Delta^{a,P}_7$ will be$(****)(****)(*000)(000*)$, when the plaintext x is randomly selected, is $2^{-6\times 8} = 2^{-48}$. The number of plaintext pairs required to obtain 2^{39} plaintext pairs with the desired property is about 2^{87}. There are $2^{128} \times 2^{2\times 8} = 2^{144}$ values of $(k_8, z_{7,2,0}, z_{7,3,3})$. Thus, the complexity of finding the "right" 3-tuples is of order $2^{39} \times 2^{144} = 2^{183}$ encryptions. The probability that particular wrong 3-tuple will be marked as wrong using only one pair of plaintexts is 2^{-32}. The number of wrong 3-tuples that are not marked as wrong after applying the procedure 2^{39} times is on average $2^{144} \times (1 - 2^{-32})^{2^{39}} \approx 2^{-40}$ i.e. the probability that only the right key will not be marked as wrong is very large. Once the right 3-tuple is determined,

Table 5.4: Impossible related-key differential attacks vs Partial sums attacks on AES-192

# of rounds	p/c pairs	Time	Attack
7	2^{111} RK-CP	2^{116}	impossible related-key differential
8	2^{88} RK-CP	2^{183}	impossible related-key differential
7	19×2^{32} CP	2^{155}	partial sums
7	$2^{128} - 2^{119}$ CP	2^{120}	partial sums
8	$2^{128} - 2^{119}$ CP	2^{188}	partial sums

it is easy to determine the rest of the key using exhaustive search. One naive way to select the set of 2^{39} plaintext pairs with desired property from the set of 2^{87} available plaintext pairs is to check whether each pair leads to the required difference $\Delta x_7^{a,P}$ for the particular key k_8. In that case, the complexity will be $2^{87+144} = 2^{231}$. The differences $\Delta x_{7,2}^{a,P}$ and $\Delta x_{7,3}^{a,P}$ depend only on eight bytes of the key k_8 and the key differences Δ_1, Δ_2 and Δ_3. Hence, a better way to select the set is to assume first the values of these eight bytes and then compute the set for every possible value of Δ_1, Δ_2 and Δ_3. Then, we can assume the rest of the key, compute the real values of Δ_1, Δ_2 and Δ_3, and select the set that corresponds to the real values of the key differences. Selection can be made by selecting those pairs such that $\Delta x_{7,3}^{a,P} = (000*)$, and then selecting the pairs that satisfy $\Delta x_{7,2}^{a,P} = (*000)$ from the previously selected pairs. The complexity in this case is about $2^{4\times 8} \times 2^{3\times 7} \times 2^{87} = 2^{140}$. Table 5.4[2] compares the complexities of impossible related-key differential attacks to the complexities of the partial sums attacks proposed in [37], which is the best attack on the 192-bit key variant known to the authors.

5.3 Is the Markov Cipher property sufficient?

The concept of Markov ciphers was introduced in order to analyze the security of iterated block ciphers against differential cryptanalysis. We give the following definition taken from [71]:

Definition 5.1 *An iterated cipher with round function $y = f(x, k)$ is* a Markov cipher *if there is a group operation for defining differences such that, for all choices of α, $\alpha \neq e$ and β, $\beta \neq e$*

$$P_o(\Delta y = \beta | \Delta x = \alpha, x = \gamma)$$

is independent of γ when the subkey k is uniformly random, where $P_o(\Delta y = \beta | \Delta x = \alpha, x = \gamma)$ is the probability when the same round key is used to encrypt γ and $\gamma + \alpha$, and e is the identity element.

[2]RK-CP stands for related-key chosen plaintext, and CP stands for chosen plaintext.

One can easily notice that *if an iterated cipher is a Markov cipher, then the previous property holds even when* $\alpha = e$ *or* $\beta = e$. The differences in the previous definition are computed when the ciphertexts are obtained using the same key. It was shown [71] that, if an iterated cipher is Markov and its round keys are independent, then the sequence of differences at each round output forms a Markov chain. Furthermore, if the Markov chain of differences has a steady state probability distribution, then this steady state distribution must be the uniform distribution. If we additionally assume that the hypothesis of stochastic equivalence holds for the Markov cipher, then, for almost all subkeys, this cipher is secure against a differential cryptanalysis attack after sufficiently many rounds (see [71] for more details).

The differences in the previous discussion are computed when the ciphertexts are obtained using the same key. In general, we can consider differences in the case when the ciphertexts are obtained using different keys. When the round keys are independent, it is obvious that we can construct highly probable related-key differentials by encrypting the same plaintext using keys that differ in one round key (the key of one of the last rounds). This is demonstrated by the following example.

Magenta [58] is 128-bit block encryption algorithm submitted for AES by Deutsche Telekom AG. It supports 128-bit, 192-bit and 256-bit key sizes. We will consider the 128-bit key variant, which consist of of six Feistel rounds. The key is divided into two 64-bit halves K_1 and K_2. The first part K_1 is used in rounds 1,2,5 and 6, and the second part K_2 is used in the remaining rounds 3 and 4. The algorithm is given by

$$E_K(M) = F_{K_1}(F_{K_1}(F_{K_2}(F_{K_2}(F_{K_1}(F_{K_1}(M)))))),$$

where

$$F_y(x) = ((x_8, \ldots, x_{15}), (x_0, \ldots, x_7) \oplus E^{(3)}(x_8, \ldots, x_{15}, y_0, \ldots, y_7)).$$

Let Δy and ΔE be two differences such that $P(\Delta E^{(3)} = \Delta E | \Delta y, \Delta x = 0)$ is significantly greater[3] than 2^{-64}. If we submit the same plaintext for encryption under the keys (K_1, K_2) and $(K_1, K_2 \oplus \Delta y)$, then the difference between the left halves at the input of the fourth round will be ΔE with probability significantly higher than 2^{-64}. We must note that, although the attack that exploits such related-key differential is more efficient than exhaustive search, the complexity of the attack is large compared to the attack proposed in [14]. It is obvious that we must take the subkey differences into account if we want to analyze the resistance of iterated ciphers to related-key differential cryptanalysis.

[3] Authors mention 2^{-40} as an upper bound for transition probabilities of $E^{(3)}$.

Definition 5.2 *We say that* the round function $y = f(x, k)$ is $\mathcal{K} - f$ *if for every* α , β *and* δ *one can find* α_1 *such that*

$$P(\Delta y = \beta | \Delta x = \alpha, \Delta k = \delta, x = \gamma) = P_o(\Delta y = \beta | \Delta x = \alpha_1, x = \gamma)$$

for any γ *and uniform distribution of the subkey* k*.*

Often, the round function is composed of key addition using bitwise XOR and bijective transformation (e.g. AES). In this case, the difference α_1 can be simply computed[4] as $\alpha_1 = \alpha \oplus \delta$. The definition of $\mathcal{K} - f$ round functions enforces relation between the probability distributions of the round output differences in the cases of zero and nonzero key differences. This is formally stated by the following theorem.

Theorem 5.1 *If the round function is* $\mathcal{K} - f$ *and the input* x *is independent of the input difference* Δx *and round key difference* Δk*, then for every* α , β *and* δ *one can find* α_1 *such that*

$$P(\Delta y = \beta | \Delta x = \alpha, \Delta k = \delta) = P_o(\Delta y = \beta | \Delta x = \alpha_1).$$

Proof.

$$\begin{aligned} & P(\Delta y = \beta | \Delta x = \alpha, \Delta k = \delta) = \\ = & \sum_{\gamma} P(\Delta y = \beta | \Delta k = \delta, \Delta x = \alpha, x = \gamma) \times P(x = \gamma) = \\ = & \sum_{\gamma} P_o(\Delta y = \beta | \Delta x = \alpha_1, x = \gamma) \times P(x = \gamma) = \\ = & P_o(\Delta y = \beta | \Delta x = \alpha_1). \end{aligned}$$

■

[4] This is the reason why we use a somewhat strange notation $\mathcal{K} - f$.

The Markov cipher property (round output difference to depend only on the the round input difference and not on the particular round inputs) is crucial in proving that the sequence of the round output differences forms a homogenous Markov chain. Therefore, it is convenient to define a similar property in the case of related-key differentials.

Definition 5.3 *An iterated cipher with round function $y = f(x, k)$ possesses* a Markov cipher property for related keys *if there is a group operation for defining differences such that, for all choices of α and β*

$$P(\Delta y = \beta | \Delta x = \alpha, x = \gamma) = P(\Delta y = \beta | \Delta x = \alpha)$$

for any probability distribution of the round key differences and uniformly distributed round key k.

The $\mathcal{K} - f$ property of the round function enables us to analyze the propagation of the related-key differences by observing the propagation of the differences when we use the same key for encryption of the pair of plaintexts. Therefore, it is not surprising that Markov ciphers with $\mathcal{K} - f$ round function possess a Markov cipher property for related keys.

Theorem 5.2 *If an iterated cipher is a Markov cipher with $\mathcal{K} - f$ round function, the round key is uniformly distributed, and the round key difference is independent of the input and the input difference, then the cipher possesses a Markov cipher property for related keys.*

Proof.

$$\begin{aligned}
& P(\Delta y = \beta | \Delta x = \alpha, x = \gamma) = \\
= & \sum_{\delta} P(\Delta y = \beta, \Delta k = \delta | \Delta x = \alpha, x = \gamma) = \\
= & \sum_{\delta} P(\Delta y = \beta | \Delta x = \alpha, \Delta k = \delta, x = \gamma) \times P(\Delta k = \delta | \Delta x = \alpha, x = \gamma) = \\
= & \sum_{\delta} P(\Delta k = \delta) \times P_o(\Delta y = \beta | \Delta x = \alpha_1, x = \gamma) \\
= & \sum_{\delta} P(\Delta k = \delta) \times P_o(\Delta y = \beta | \Delta x = \alpha_1) \\
= & \sum_{\delta} P(\Delta k = \delta) \times P(\Delta y = \beta | \Delta x = \alpha, \Delta k = \delta) \\
= & \sum_{\delta} P(\Delta k = \delta, \Delta y = \beta | \Delta x = \alpha) \\
= & P(\Delta y = \beta | \Delta x = \alpha).
\end{aligned}$$

■

The previous results provide intuition for proving the following theorem.

Theorem 5.3 *If* (i) *an r-round iterated cipher is a Markov cipher with $\mathcal{K} - f$ round function,* (ii) *the round keys k_i are independent and uniformly random and* (iii) *the round key differences are independent random variables, then the sequence of differences $\Delta x = \Delta y(0), \Delta y(1), \ldots \Delta y(r)$ is a Markov chain. If additionally* (iv) *the probability distributions $p(\Delta k_i)$ are identical, then the Markov chain is homogeneous.*

Proof.

$$
\begin{aligned}
& P(\Delta y(i) = \beta_i | \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) = \\
= & \sum_{\gamma} P(\Delta y(i) = \beta_i, y(i-1) = \gamma | \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) = \\
= & \sum_{\gamma} P(\Delta y(i) = \beta_i | y(i-1) = \gamma, \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) \times \\
& P(y(i-1) = \gamma | \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) = \\
= & \sum_{\gamma} P(\Delta y(i) = \beta_i | y(i-1) = \gamma, \Delta y(i-1) = \beta_{i-1}) \times \\
& P(y(i-1) = \gamma | \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) = \\
= & \sum_{\gamma} P(\Delta y(i) = \beta_i | \Delta y(i-1) = \beta_{i-1}) \times \\
& P(y(i-1) = \gamma | \Delta y(i-1) = \beta_{i-1}, \ldots, \Delta x = \alpha) = \\
= & P(\Delta y(i) = \beta_i | \Delta y(i-1) = \beta_{i-1})
\end{aligned}
$$

If the probability distributions $P(\Delta k_i)$ are identical, then

$$
\begin{aligned}
& P(\Delta y(i) = \beta | \Delta y(i-1) = \alpha) = \\
= & \sum_{\delta} P(\Delta y(i) = \beta, \Delta k_i = \delta | \Delta y(i-1) = \alpha) = \\
= & \sum_{\delta} P(\Delta y(i) = \beta | \Delta k_i = \delta, \Delta y(i-1) = \alpha) \times P(\Delta k_i = \delta) = \\
= & \sum_{\delta} P_o(\Delta y = \beta | \Delta x = \alpha_1) \times P(\Delta k_i = \delta) = \\
= & \sum_{\delta} P_o(\Delta y = \beta | \Delta x = \alpha_1) \times P(\Delta k_{i-1} = \delta) = \\
= & P(\Delta y(i-1) = \beta | \Delta y(i-2) = \alpha).
\end{aligned}
$$

■

Now, suppose that the round keys and round key differences are independent and uniformly distributed. If the round input difference is uniformly distributed, then the round output difference is also uniformly distributed. Hence, if the Markov chain formed by the round output differences has steady-state probability distribution, then this steady-state distribution must be the uniform distribution. Usually, the round keys are derived using some key generation algorithm, and, given the key and the key difference, the round keys and the round key differences are uniquely determined. If we assume that the probability of the related-key differentials when the round keys and round key differences are fixed, and the probability of the related-key differentials when the round keys and round key differences are independent and uniformly distributed are approximately equal, then the previously discussed Markov ciphers are secure against related-key differential attack after sufficiently many rounds. We will refer to this assumption as *hypothesis of stochastic equivalence for related keys*.

The previous discussion suggests that one way of dealing with related-key differential cryptanalysis is to use key scheduling algorithms whose output is "close" to random. We already mentioned that the success of a related-key attack depends on the attacker's ability to find highly probable (or impossible) related-key differential trails. Unpredictable key differences make the task of constructing such related-key differential trails very difficult.

REFERENCES

[1] V. Afanassiev, C. Gehrmann and B. Smeets, "Fast Message Authentication Using Efficient Polynomial Evaluation," Proceedings of Fast Software Encryption Workshop 1997, pp. 190-204. 1.3, 2.6.3

[2] W. Alexi, B. Chor, O. Goldreich, C.P. Schnor, "RSA/Rabin Functions: Certain Parts Are as Hard as the Whole," SIAM Journal on Computing, Vol. 17, pp. 194-209, 1988. 1.3

[3] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham,"A New Family of Authentication Protocols," ACM Operating Systems Review 32(4), pp. 9-20, 1998. 1.3, 4

[4] M. Bellare, O. Goldreich, S. Goldwasser, "Incremental Cryptography: The case of Hashing and Signing," Crypto'94, LNCS 839, Springer-Verlag, pp. 216-233. 1.3

[5] M. Bellare, J. Killian, P. Rogaway,"The Security of Cipher Block Chaining Message Authentication Code," Advances in Cryptology - Proceedings of Crypto'94, Lecture Notes in Computer Science, Vol. 839, pp. 341-358, Springer-Verlag, 1994. 1.3, 3.1

[6] M. Bellare, R. Guerin and P. Rogaway, "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions," Advances in Cryptology - Proceedings of Crypto'95, Lecture Notes in Computer Science, Vol. 963, pp. 15-29, Springer-Verlag, 1995. 1.3, 3.2.4, 3.3

[7] E.R. Berlekamp, R.J. McEliece, H.C.A. van Tilborg, "On the Inherent Intractability of Certain Coding Problems," IEEE Transactions on Information Theory, 1978. 1.3

[8] A. Berendschot, B. den Boer, J. P. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgard, M. Dichtl, W. Fumy, M. van der Ham, C. J. A. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, and J. Vandewalle, "Final Report of RACE Integrity Primitives," LNCS 1007, Springer-Verlag, 1995. 1.3

[9] F. Bergadano, D. Cavagnino, B. Crispo, "Chained Stream Authentication," Proceeding of Selected Areas in Cryptography 2000, pp. 142-155, 2000. 1.3, 4

[10] J. Bierbrauer, T. Johansson, G. Kabatianskii and B. Smeets, "On Families of Hash Functions Via Geometric Codes and Concatenation," Proceedings of CRYPTO '93, Springer-Verlag, pp. 331-342, 1994. 1.3, 2.7

[11] E.Biham and A.Shamir, "Differential cryptanalysis of DES-like cryptosystems," Journal of Cryptology, 4(1):3-72, 1991. 1.3

[12] E. Biham and A. Shamir, 'Differential Cryptanalysis of Snefru, Khafre, REDOC II, LOKI, and Lucifer," Advances in Cryptology, CRYPTO '91 Proceedings, Springer- Verlag, 1992, pp. 156-171. 1.3

[13] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," Journal of Cryptology, v. 7, n. 4, 1994, pp. 229-246. 1.3, 4.3.1

[14] E. Biham, A. Biryukov, N. Ferguson, L. Knudsen, B. Schneier, A. Shamir, "Cryptanalysis of MAGENTA", http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm 5.3

[15] E.Biham and N.Keller, "Cryptanalysis of Reduced Variants of Rijndael," http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html 5.2.3

[16] J. Black and P. Rogaway, "CBC MACs for Arbitrary-Length Messages: The Three-Key Construction," Advances in Cryptology - Proceedings of Crypto 2000, Lecture Notes in Computer Science, vol. 1880, pp. 197-215, Springer-Verlag, 2000. 1.3, 3.3

[17] J. Black and P. Rogaway, "A Block Cipher Mode of Operation for Parallelizable Message Authentication," Advances in Cryptology - Proceedings of Eurocrypt 2002, Lecture Notes in Computer Science, vol. 2332, pp. 384-397, Springer-Verlag, 2002. 1.3, 3.3, 3.4.3

[18] D. Bleichenbacher and U. Maurer,"On the Efficiency of One-Time Digital Signatures," Advances in Cryptology - Proceedings of AsiaCrypt '96, LNCS 1163, Springer-Verlag, 1996. 1.3

[19] L. Blum, M. Blum, M. Shub, "A Simple Secure Unpredictable Pseudo-Random Number Generator," SIAM Journal on Computing, Vol. 15, pp. 364-383, 1986. 1.3

[20] M. Blum, S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudorandom Bits," SIAM Journal on Computing, Vol. 13, pp. 850-864, 1984. 1.3

[21] M. Blum, S. Goldwasser, "An Efficient Probabilistic Public-Key Encryption Scheme which hides all partial information," Crypto'84, LNCS 196, Springer-Verlag, pp. 289-302.

[22] M. Burrows, M. Abadi and R. Needham,"A Logic of Authentication," DEC SRC Research Report 39. 3.3

[23] R. Canneti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," In Infocom '99, 1999. 4

[24] E. Carrara and M. Baugher, "The Use of TESLA in SRTP," Internet draft, `http://ietfreport.isoc.org/ids-wg-msec.html`. 4

[25] J.L. Carter and M.N. Wegman, "Universal Classes of Hash Functions," Journal of Computer and System Sciences, Vol. 18, pp. 143-154, 1979. 1.3

[26] A. Chan, "A graph-theoretical analysis of multicast authentication," Proc. of the 23rd Int. Conf. on Distributed Computing Systems, 2003.

[27] J. Cheon, M. Kim, K. Kim, J. Lee, and S. Kang, "Improved Impossible Differential Cryptanalysis of Rijndael and Crypton," Information Security and Cryptology - ICISC 2001 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings, LNCS 2288,p. 39 ff. 5.2.3

[28] S. Cheung,"An Efficient Message Authentication Scheme for Link State Routing," Proceedings of the 13th Annual Computer Security Application Conference, 1997. 1.3, 4

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms," Second edition, McGraw-Hill, 2001.

[30] J.Daemen and V.Rijmen, "AES Proposal: Rijndael," http://csrc.nist.gov/encryption/aes. 1.3, 5.2.1

[31] J.Daemen, "Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis," Doctoral Dissertation, March 1995, K.U.Leuven. 1.3, 5.2.1

[32] I. Damgard, "Collision Free Hash Functions and Public Key Signature Schemes," Eurocrypt'87, LNCS 304, Springer-Verlag, pp. 203-216. 1.3

[33] Y. Desmedt, Y. Frankel and M. Yung, "Multi-Receiver/Multi-Sender Network Security: Efficient Authenticated Multicast/Feedback," INFOCOM, 1992, pp.2045-2054. 1.3

[34] Y. Desmedt and K. Kurosawa,"How to Break a Practical MIX and Design a New One," Eurocrypt 2000, LNCS 1807, Springer-Verlag, pp. 557-572. 6

[35] W. Diffie, M.E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, IT-22 (Nov.), pp.644-654, 1976. 1.3

[36] N. Ferguson and B. Schneier, "A Cryptographic Evaluation of IPsec," technical report, 2000.

[37] N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, D. Whiting, "Improved Cryptanalysis of Rijndael," 7th International Workshop, FSE 2000, New York, NY, USA, April 2000, Proceedings, LNCS 1978, p. 213 ff. 5.2.3

[38] FIPS PUB 113, Computer Data Authentication. 1.3

[39] FIPS PUB 186, Digital Signature Standard. 1.3

[40] FIPS PUB 180-2, Secure Hash Standard.

[41] FIPS PUB 197, Advanced Encryption Standard (AES). 1.3, 4.1.2, 5.2

[42] FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC). 1.3, 4.1.4

[43] R. Gennaro and P. Rohatgi,"How to Sign Digital Streams," Advances in Cryptology - Proceedings of Crypto '97, LNCS 1294, Springer-Verlag, 1997, pp. 180-197. 1.3, 3.1, 4

[44] E.N. Gilbert, F.J. MacWilliams, N.J.A. Sloane, "Codes which detect deception," Bell Sys. Tech. J., Vol. 53, pp. 405-424, 1974.

[45] O. Goldreich, S. Goldwasser and S. Micali,"How to Construct Random Functions," Journal of the ACM, Vol.33, No.4, 210-217, 1986. 1.3

[46] O. Goldreich, H. Krawcyzk, M. Luby, "On the existence of Pseudorandom Generators," SIAM Journal on Computing, Vol.22, No.6, pp. 1163-1175, 1993. 1.3

[47] O. Goldreich,"Foundations of Cryptography: Basic Tools," Cambridge University press, 2001. 4.1.2

[48] O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman,"Security Preserving Amplification of Hardness," 31st IEEE Symposium on Foundations of Computer Science, pp. 318-326, 1990. 1.3

[49] O. Goldreich, S. Goldwasser, S. Micali, "On the Cryptographic Applications of Random Functions," Crypto'84, LNCS 263, Springer-Verlag, pp. 276-288, 1985. 1.3

[50] S. Goldwasser, S. Micali, "Probabilistic Encryption," Journal of Computer and System Science, Vol. 28, No. 2, pp. 270-299, 1984.

[51] S. Goldwasser, S. Micali, P. Tong, "Why and How to Establish a Private Code in a Public Network," 23rd Symposium on Foundations of Computer Science, pp. 134-144, 1982.

[52] S. Goldwasser, S. Micali, A.C. Yao, "Strong Signature Schemes," 15th ACM Symposium on the Theory of Computing, pp.431-439, 1983. 1.3

[53] S. Goldwasser, S. Micali, and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," SIAM Journal on Computing, 17(2):281-308, April 1988. 1.3, 3, 3.1

[54] J. Hastad, R. Impagliazzo, L.A. Levin, M. Luby, "Construction of a Pseudorandom Generator from Any One-Way Function," SIAM Journal on Computing, Vol. 28, No. 4, pp. 1364-1396, 1999. (Preliminary version by Impagliazzo et al. in 21st ACM Symposium on the Theory of Computing (1989) and Hastad in 22nd ACM Symposium on the Theory of Computing (1990).) 1.3

[55] R. Impagliazzo, M. Naor, "Efficient Cryptographic Schemes Provable as Secure as Subset Sum," Journal of Cryptology, Vol. 9, pp. 199-216, 1996. 1.3

[56] R. Impagliazzo, L. Levin and M. Luby,"Pseudo-Random Generation from One-Way Functions," Proceedings of the Twenty First Annual Symposium on the Theory of Computing, ACM, 1989.

[57] T. Iwata and K. Kurosawa,"OMAC: One-Key CBC MAC," Fast Software Encryption, FSE 2003, LNCS 2887,pp.129-153, Springer,2003. 1.3, 3.2.4, 3.3, 3.4.3

[58] M.J. Jacobson,Jr and K. Huber, "The MAGENTA Block Cipher Algorithm," AES candidate, http://csrc.nist.gov/encryption/aes. 5.3

[59] G. Jakimoski, "Unconditionally Secure Information Authentication in Presence of Erasures," Proceedings of the 10th IMA International Conference on Cryptography and Coding, LNCS 3796, pp. 304-321, 2005. 1

[60] G. Jakimoski and Y. Desmedt, "Related-key Differential Cryptanalysis of 192-bit Key AES Variants," Proceedings of the 10th Workshop on Selected Areas of Cryptography, LNCS 3006, pp. 208-221, Springer, 2004. 4.3.1, 1

[61] E. Jaulmes, A. Joux and F. Valette,"On the Security of Randomized CBC-MAC Beyond the Birthday Paradox Limit: A New Construction," Fast Software Encryption, FSE 2002, LNCS 2365, pp. 237-251, Springer-Verlag. 1.3

[62] T. Johansson, G. Kabatianskii and B. Smeets, "On the Relation Between A-codes and Codes Correcting Independent Errors," Proceedings of EUROCRYPT 1993, Springer-Verlag, pp. 1-11, 1994. 1.3

[63] A. Joux, G. Martinet, and F. Valette, "Blockwise-Adaptive Attackers Revisiting the (In)Security of Some Provably Secure Encryption Modes: CBC, GEM, IACBC," Advances in Cryptology - Proceedings of Crypto 2002, Lecture Notes in Computer Science, vol. 2442, pp. 17-30, Springer-Verlag, 2002.

[64] C. Kaufman, R. Perlman and M. Speciner,"Network Security: Private Communication in a Public World," Prentice Hall, 2002.

[65] J. Kelsey, B. Schneier and D. Wagner, "Key-schedule cryptanalysis of IDEA, GDES, GOST, SAFER, and Triple-DES," Advances in Cryptology, Proceedings Crypto'96, LNCS 1109, pp.237-252. 1.3

[66] J. Kelsey, B. Schneier and D. Wagner, "Related-key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2 and TEA," Proceedings of ICICS'97, pp. 233-246, Springer-Verlag, 1997. 4.3.1

[67] L.R. Knudsen, "Truncated and Higher Order Differentials," Fast Software Encryption, 2nd International Workshop Proceedings, Springer-Verlag, 1995, pp. 196-211. 1.3, 5.2.2

[68] H. Krawczyk, "LFSR-based Hashing and Authentication," Crypto'94, LNCS 839, Springer-Verlag, pp. 129-139.

[69] H. Krawczyk, "New Hash Functions For Message Authentication," Eurocrypt'95, LNCS 921, Springer-Verlag, pp. 301-310.

[70] X. Lai, "Higher Order Derivations and Differential Cryptanalysis," Communications and Cryptography: Two Sides of One Tapestry, Kluwer Academic Publishers, 1994, pp. 227-233. 1.3

[71] X. Lai, J. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," Advances in Cryptology, CRYPTO '91 Proceedings, Springer-Verlag, 1991, pp. 17-38. 1.3, 5.3, 5.3

[72] L. Lamport,"Constructing Digital Signatures From a One-Way Function," no. CSL 98, 1979. 1.3

[73] L.A. Levin, "One-Way Functions and Pseudorandom Generators," Combinatorica, Vol. 7, pp. 357-363, 1987. 1.3

[74] M. Luby, C. Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Functions," SIAM Journal on Computing, Vol. 17, pp. 373-386, 1988. 1.3

[75] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-Resilient Codes", Proc. 29 th Symp. on Theory of Computing, 1997, pp. 150-159.

[76] J.L. Massey, "Contemporary Cryptology: An Introduction,"in Contemporary Cryptology, The Science of Information Integrity, ed. G.J. Simmons, IEEE Press, New York, 1992. 2.1

[77] M. Matsui, "Linear cryptanalysis method for DES cipher," In Advances in Cryptology - EUROCRYPT'93, LNCS 765, pp. 386-397, Springer-Verlag, 1993. 1.3

[78] A.J. Menezes, P.C. van Oorschot,S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.

[79] R.C. Merkle, M.E. Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks," IEEE Transactions on Information Theory, Vol. 24, pp. 525-530, 1978. 1.3

[80] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," Advances in Cryptology - Proceedings of Crypto '87, LNCS 293, Springer-Verlag, 1987, pp. 369-378. 1.3

[81] S. Micali, C. Rackoff, B. Sloan, "The Notion of Security for Probabilistic Cryptosystems," SIAM Journal on Computing, Vol. 17, pp. 412-426, 1988.

[82] W.H.Mills,"Covering design I: coverings by a small number of subsets," Ars Combin. 8, pp. 199-315, 1979. 2.6.1

[83] S. Miner and J. Staddon, "Graph-Based Authentication of Digital Streams," The 2001 IEEE Symposium on Security and Privacy. 1.3

[84] C.Mitchell and M.Walker,"Solutions to the Multidestination Secure Electronic Mail Problem," Computers and Security, Vol.7(1988), pp.483-488.

[85] M. Naor, M. Yung, "Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks," 22nd ACM Symposium on the Theory of Computing, pp. 427-437, 1990.

[86] M. Naor, M. Yung, "Universal One-Way Hash Functions and their Cryptographic Applications," 31st ACM Symposium on the Theory of Computing, pp.33-43, 1989.

[87] J.M. Park, E.K.P. Chong and H.J. Siegel, "Efficient Multicast Stream Authentication Using Erasure Codes," ACM Transactions on Information and System Security, Vol. 6, No. 2, May 2003, pp. 258-285.

[88] A. Perrig, R. Canneti, J. D. Tygar, D. Song, "Efficient Authentication and Signing of Multicast Streams Over Lossy Channels," Proceedings of the IEEE Security and Privacy Symposium, 2000. 1.1.5, 1.1.5, 1.3, 4, 4.1.2, 4.4

[89] A. Perrig, R. Canneti, D. Song and J. D. Tygar , "Efficient and Secure Source Authentication for Multicast," Proceedings of the Network and Distributed System Security Symposium, 2001. 4

[90] A. Perrig, R. Canneti, J. D. Tygar and D. Song, "The TESLA Broadcast Authentication Protocol," RSA CryptoBytes, Volume 5, No.2, 2002. 4

[91] A. Perrig and J. D. Tygar, "Secure Broadcast Communication in Wired and Wireless Networks," Kluwer Academic Publishers, 2002. 4

[92] E. Petrank and C. Rackoff,"CBC MAC for Real-Time Data Sources," J. Cryptology, vol.13, no.3, pp.315-338, Springer-Verlag, 2000. 1.3

[93] J.J. Quisquater and D. Samyde, "Eddy current for Magnetic Analysis with Active Sensor", Proceedings of Esmart 2002 3rd edition, Nice, France, September 2002.

[94] M.O. Rabin, "Digitalized Signatures and Public Key Functions as Intractable as Factoring," TR-212, LCS, MIT, Cambridge, MA, 1979. 1.3

[95] M. Rabin,"Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," J. ACM 36, 2, pp.335-348.

[96] R.Rees, D.R.Stinson, R.Wei and G.H.J. van Rees, "An application of covering designs: Determining the maximum consistent set of shares in a threshold scheme," Ars Combin. 531, pp. 225-237, 1999. 2.6.1

[97] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," CACM, Vol. 21, pp. 120-126, 1978. 1.3

[98] R.L. Rivest, "The MD5 message digest algorithm," Internet Request for Comments, April 1992, RFC 1321. 4.1.4

[99] P. Rohatgi, "A compact and fast hybrid signature scheme for multicast packet authentication," In 6th ACM Conference on Computer and Communications Security, November 1999. 4

[100] A. D. Rubin and P. Honeyman,"Formal Methods for the Analysis of Authentication Protocols," CITI Technical Report 93-7, 1993. 3.3

[101] C. Schnor,"Efficient Identification and Signatures for Smart Cards," Advances in Cryptology - Crypto '89 Proceedings, LNCS 435, Springer-Verlag, 1989. 1.3

[102] G.J. Simmons,"Authentication Theory / Coding Theory," Proceedings of CRYPTO '84, LNCS 196 (1985), pp. 411-432. 1.3, 2.1, 2.2, 2.2

[103] G.J. Simmons, " A Survey of Information Authentication," in Contemporary Cryptology, The Science of Information Integrity, ed. G.J. Simmons, IEEE Press, New York, 1992. 1.3, 2.1

[104] C.E. Shannon, "Communication Theory of Secrecy Systems," Bell Sys. Tech. J., Vol. 28, pp. 656-715, 1949.

[105] D.R. Stinson,"Cryptography: Theory and Practice," CRC Press, 1995.

[106] D.R. Stinson, "Some Constructions and Bounds for Authentication Codes," Journal of Cryptology 1 (1988), pp. 37-51. 1.3, 2.5

[107] D.R. Stinson, "The Combinatorics of Authentication and Secrecy Codes," Journal of Cryptology 2 (1990), pp. 23-49. 1.3, 2.5

[108] D.R. Stinson, "Combinatorial Characterizations of Authentication Codes," Proceedings of CRYPTO '91, LNCS 576 (1992), pp.62-73. 1.3, 2.5

[109] D.R. Stinson, "Universal Hashing and Authentication Codes," Proceedings of CRYPTO '91, LNCS 576 (1992), pp. 74-85. 1.3

[110] D.R. Stinson, R. Wei and L. Zhu, "Some new bounds for cover-free families," J. Combin. Theory A. 90 (2000), pp. 224-234. 2.1, 2.6.2

[111] P.F. Syverson, S.G. Stubblebine and D.M.Goldschlag, "Unlinkable serial transactions," In Financial Cryptography '97, Springer Verlag, LNCS 1318, 1997. 4

[112] U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," 25th Symposium on Foundations of Computer Science, pp. 458-463. 1.3

[113] X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, "Cryptanalysis for Hash Functions MD4 and RIPEMD," Proceedings of Eurocrypt '05, LNCS 3494, pp. 1-18, Springer, 2005. 4.1.4

[114] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," Proceeding of Eurocrypt '05, LNCS 3494, pp. 19-35, Springer, 2005. 4.1.4

[115] M.N. Wegman and J.L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality," Journal of Computer and System Sciences, Vol. 22, pp. 265-279, 1981. 1.3

[116] C. K. Wong, S. S. Lam, "Digital Signatures for Flaws and Multicasts," Proceedings of IEEE ICNP '98, 1998. 1.3, 4

[117] A.C. Yao,"Theory and Application of Trapdoor Functions," 23rd IEEE Symposium on Foundations of Computer Science, pp. 80-91, 1982. 1.3

[118] K. Zhang,"Efficient Protocols for Signing Routing Messages," Proceedings of the Symposium on Network and Distributed System Security, 1998. 1.3, 4

BIOGRAPHICAL SKETCH

Goce C. Jakimoski

Goce C. Jakimoski has completed his Bachelors degree in Electrical Engineering, Electronics and Telecommunications, at Ss. Cyril and Methodius University, Skopje, Macedonia, in July 1995. He has received a Masters degree in Electrical Engineering from the same university in July 1998 under the advisement of Prof. Ljupco Kocarev. From January 1996 to July 2001, Goce has worked as an electrical engineer. He enrolled in the doctoral program at the Department of Computer Science, Florida State University, in the fall of 2001.

Goce's research interests include cryptography, computer and network security, and he is author or co-author of a number of papers in these areas.